AD_____

Award Number: DAMD17-99-1-9356

TITLE: Innovative Statistical Approaches to Modeling Multiple Data from the NSABP BCPT

PRINCIPAL INVESTIGATOR: Lisa A. Weissfeld, Ph.D.
Kiros Berhane, Ph.D.

CONTRACTING ORGANIZATION: University of Pittsburgh
Pittsburgh, Pennsylvania 15261

REPORT DATE: September 2001

TYPE OF REPORT: Annual

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

Table of Contents

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2001	Annual (01 Sep 00 - 31 Aug 01)

4. TITLE AND SUBTITLE
Innovative Statistical Approaches to Modeling Multiple Outcome Data from the NSABP BCPT

5. FUNDING NUMBERS
DAMD17-99-1-9356

6. AUTHOR(S)
Lisa A. Weissfeld, Ph.D.
Kiros Berhane, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pittsburgh
Pittsburgh, Pennsylvania 15261

E-Mail: lweis@pitt.edu

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The goals of this IDEA award are two-fold, namely, to develop and refine statistical methodology for the analysis of a data set where multiple outcomes or disease incidence endpoints are of interest and to apply these methods to the analysis of a data set in prevention such as the National Surgical Adjuvant Breast and Bowel Project's Breast Cancer Prevention Trial. The software for the fitting of the B-spline models, the first type of model that was proposed in this study is now being used to fit these models and the simulation studies are also complete. Software development for the pseudospline approach is nearly complete. Extensions to the analysis of recurrent event data outcomes are now being developed.

14. SUBJECT TERMS
Breast Cancer, Statistical Methodology, Survival Analysis

15. NUMBER OF PAGES
196

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

INTRODUCTION

This goal of this work is to extend the current statistical methodology for the analysis of multiple outcome data, to include a more flexible class of modeling techniques. The method proposed by Wei, Lin, and Weissfeld (1989) has been found to be an extremely flexible and useful statistical method for the analysis of multiple outcome data. However, this method suffers from the limitation that each outcome must follow the proportional hazards model. In the extension of the methodology that is developed through the work on this grant, we present an extension of the method of Wei, Lin and Weissfeld (1989) that incorporates a spline based version of the Cox proportional hazards model that is proposed by Gray (1992). The advantage of this approach is that the proportionality assumption is not needed for the modeling of each outcome. In fitting this model to the outcome data, one obtains a more detailed view of the relationship between failure time and the given covariates of interest.

BODY:

The work included in the statement of work involves several components, the development of flexible marginal models for multiple time to event data using penalized B-spline based models, to extend these models using pseudosplines, and the development of regression diagnostics and goodness-of-fit tests for these models. Substantial progress has now been made on several of these aims.

The investigators, Dr. Kiros Berhane and Dr. Lisa Weissfeld, are at the University of Sourthern California and the University of Pittsburgh, respectively. There is a graduate student research at the University of Pittsburgh site who works closely with both Dr. Berhane and Dr. Weissfeld. This individual, Zekarias Berhane has been working on the grant since its beginning. He is now very experienced in the use of the software that is needed and has been instrumental in its development. There has been one meeting over the past year. This meeting took place in August at the Joint Statistical Meetings in Atlanta, GA. Dr. Berhane, Mr. Berhane and Dr. Weissfeld were all present at this meeting and spent approximately 20 hours together during this time period working on the software and refining the proposed methodology.

Throughout much of the past year the work on the grant has required a substantial portion of the investigators' time. There have been several problems in the implementation of the methodology that turned out to be more difficult to solve than was initially anticipated. These problems centered around the testing of the methodology, the estimation of the "meat" of the sandwich estimator for the variance covariance matrix of the proposed estimator, and the distribution of the test statistic needed for drawing inferences based on the model. A second Ph.D. student of Dr. Weissfeld, Mr. Zdenek Valenta has also been working on parts of the grant as a topic for his Ph. D. dissertation.

Specific Aim 1:

The goal of this aim is to develop flexible marginal models for multiple time to event data using penalized B-spline based models. We experienced several problems related to the completion of this aim over the past year. These problems centered on the refinement of the software and the methodology. We had to solve two major problems with respect to the methodology development over the past year: the estimator of the meat of the sandwich for the variance

covariance matrix and the form of the test statistic used for inferences drawn from the model. Based on the simulation results that are presented in the attached paper, the form of the estimator is now correct. As part of this development, we also relied on joint work that Dr. Weissfeld was doing with Mr. Valenta as part of his Ph.D. dissertation. This work is used in justifying the form of the variance covariance matrix as given in this paper. The goal of this work was the development of an appropriate survival estimator for Gray's model. This estimator is then used to estimate the cumulative hazard function, which is needed for the meat of the sandwich estimator. This work will appear in Statistics in Medicine, was presented at the ENAR meetings in Charlotte, NC in March of 2001 winning a student travel award. The second problem encountered in the development of the methodology is the derivation of the appropriate test statistic for forming inferences based on the model. This problem is now also solved. The software is now developed and running so that the method can be executed.

The major goal of this specific aim is now virtually complete and we are putting the final touches on the paper. In addition to the work initially proposed for this aim, there have been several additional pieces of work that are currently underway:

a. the extension of the method of Wei, Lin and Weissfeld (1989) and Andersen and Gill (1982) for the modeling of recurrent event data. This work is being done by Zekarias Berhane as part of his Ph.D. dissertation and will be presented at the ENAR meetings in March 2002. This is potentially important work since the WLW method based on the Cox proportional hazards model does not perform well for the modeling of recurrent data. This poor performance is due to the lack of proportional hazards in the margins causing the Cox based approach to break down. Use of Gray's model for the margins should improve on this method.
b. The development of an estimator for the survival curve. This piece of work is needed for the meat of the sandwich estimator. This work is complete and will appear in Statistics in Medicine.
c. An examination of the power of the WLW method based on Gray's model. Some preliminary work has been done for this piece and may form the basis for a future grant.
d. Inclusion of the model with time-varying coefficients. This work is analogous to that done for aim 1.

Specific Aim 2:

The goal of this aim is to develop flexible marginal models for multiple time to event data using pseudospline based models for the time to event data. This piece of work was delayed due to the problems encountered in implementing Aim 1. While preliminary software development and a draft of a manuscript are underway, much of this work was held up by the problems encountered in developing Aim 1. Since these problems have now been fixed, the work on Aim 2 will proceed in a timely fashion and should be completed in the very near future. Many of the issues discussed in Aim 1 are also important for the pseudospline based model.

Note that this work is now very close to completion.

Specific Aim 3:

We have begun some work on this aim; however, because of the work of aim 1, we have had little time to pursue this avenue of research. Mr. Berhane has met with Dr. Chang, who developed the projected and recursive residuals for the Cox model. As details are wrapped up from Aim 1, this work will take over part of Mr. Berhane's time as the GSR.

Specific Aim 4:

This work is currently underway and will serve as the second paper for Mr. Valenta's dissertation. Mr. Valenta is currently working on the extension of several different approaches for application to Gray's model.

KEY RESEARCH ACCOMPLISHMENTS:

The key research accomplishments to data from this work are:

- A program for running the multiple outcomes model based on the spline-based version of Cox's model as proposed by Gray.
- A preliminary version of a program for the multiple outcomes model based on the pseudo-spline based model.
- Software to run simulations for aim 1. The results of the simulation study are in the attached manuscript "Modeling Multiple Time-to-Event Data Using Penalized B-splines.
- Submission of the manuscript "Modeling Multiple Time-to-Event Data Using Penalized B-splines" to the NSABP group for preliminary approval before submission to a journal.
- Publication of the manuscript "Estimation of the Survival Function for Gray's Piecewise-Constant Time-Varying Coefficients Model" in Statistics in Medicine.
- Preliminary development of software for the modeling of recurrent event data using the approach of Andersen and Gill (1982).

REPORTABLE OUTCOMES:

- Attached manuscript "Modeling Multiple Time-to-Event Data Using Penalized B-splines" which has been submitted for internal NSABP review.
- Attached manuscript "Estimation of the Survival Function for Gray's Piecewise-Constant Time-Varying Coefficients Model" which is to appear in Statistics in Medicine.
- Presentation of "Flexible Models for Multiple Time-to-Event Data Using Penalized B-Splines" at the Joint Statistical Meetings in Atlanta in August 2001.
- Attached program for computing the estimators presented in the manuscript "Modeling Multiple Time-to-Event Data Using Penalized B-splines".

CONCLUSIONS:

This work provides researchers with another tool for the analysis of multiple outcome survival data. The advantage of this work is that the underlying modeling technique allows for greater flexibility when specifying the relationship between time to event and a given covariate. This is particularly applicable for the risk stratification variable used in the NSABP BCPT. For this variable the level of risk is quite different for individuals with a risk score of 10 or greater versus individuals with a risk score of less than 10. This illustrates the potential usefulness of this approach for the analysis of survival data. The analysis of the multiple outcomes verifies the fact that endometrial cancer is a significant side effect for women using tamoxifen for breast cancer prevention.

The work on Aim 1 for the grant is essentially complete. However, this work has lead to many new ideas that are being pursued through other venues. For example, the graduate student researcher, Zekarias Berhane, will be examining extensions to the recurrent event problem based on this work. Mr. Valenta also completed work on a survival function estimator that was used for formulating the variance-covariance estimator for the WLW extension. We will also spend time examining the properties of the proposed test statistics under various scenarios, focusing on power. The work for Aim 2 is now well along. Aim 4 will serve as a dissertation topic for Mr. Valenta. He is currently pursuing goodness-of-fit methodology for Gray's model and we are in the process of identifying the appropriate approach to this problem. We will be pursuing the residual analysis in the coming months.

REFERENCES:

Gray, R. J. (1992). Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. *J. Amer. Statist. Assoc.* ***87,*** 942-950.

Wei, L. J., Lin, D. Y. and Weissfeld, L. A. (1989). Regression analysis of multivariate incomplete failure time data by modeling marginal distributions. *J. Amer. Statist. Assoc.* ***84,*** 1065-1073.

APPENDIX 1. COPY OF "MODELING MULTIPLE TIME-TO-EVENT DATA USING PENALIZED B-SPLINES"

Modeling Multiple Time-to-Event Data Using Penalized B-splines

Kiros Berhane and Lisa A. Weissfeld *

---

* Kiros Berhane is Assistant Professor, Department of Preventive Medicine, University of Southern California, 1540 Alcazar Street CHP-220, Los Angeles, CA. Lisa A. Weissfeld is Professor, Department of Biostatistics, University of Pittsburgh, 303 Parran Hall, Pittsburgh, PA 15261.

Abstract

Penalized B-splines have been applied to time-to-event data, providing an extension of the proportional hazards model for a single outcome (Gray, 1994). We use this technique to extend the marginal models of Wei, Lin and Weissfeld (1989). This allows for greater flexibility in modeling the margins and makes formal development of inferential procedures possible. Applications to data from the NSABP-BCPT on the effectiveness of the drug Tamoxifen as a prevention tool against breast cancer will be discussed in detail. Results from extensive simulation studies on the small sample properties of the asymptotic tests will also be presented.

KEY WORDS: Survival analysis; Smoothing; Ridge regression; Additive models; Splines; Proportional hazards.

1 Introduction

The advent of promising drugs like tamoxifen in the treatment and/or prevention of breast cancer has ignited both hope and controversy in the scientific world and the general public. The controversy revolves around the issue of whether the benefits of the drug offset its known adverse side effects. One of the main studies that has been conducted to study the effectiveness of tamoxifen as a preventive agent for breast cancer is the Breast Cancer Prevention trial, hereafter referred to as BCPT (Fisher et al, 1998). It has been shown that tamoxifen, when used for at least 5 years, was effective in prolonging disease free survival and in reducing the rate of recurrences of second primary tumors in contralateral breast and ipsilateral breast tumor. It has also been shown that tamoxifen reduces the risk of invasive breast cancer in women that are at elevated risk due to various factors. But, there is also evidence that use of tamoxifen is positively associated with invasive endometrial cancer, ischemic heart disease, transient ischemic attack, deep vein thrombosis and/or pulmonary embolism. In order to demonstrate the positive or negative effectiveness of tamoxifen, one needs to compare the advantages of the drug to its disadvantages in a simultaneous and comprehensive manner. To do this, one needs to be able to make simultaneous inference on several time-to-event outcomes and also be able to flexibly model the effect of risk and/or prognostic factors that have non-linear effects. Considerable progress has been made over the years in the development of models that handle multiple time-to-event outcome data and models that allow for flexible modeling of effects of prognostic factors for single time-to-event outcome. But, to date, flexible methods do not exist that allow for simultaneous inference of multiple time-to-event outcomes. In this paper, we develop new inferential methods that allow for simultaneous inference on flexible models for multiple time-to-event outcomes.

The proportional hazards model (Cox 1972) has received considerable attention as a

popular way of modeling, possibly censored, time-to-event data. In addition to the proportionality of the hazards, the model assumes that the effects of the predictors (risk factors) on the response follow a parametric (mostly linear) form. Recently, this assumption has been relaxed to allow for data-dependent, and possibly non-linear, covariate effects by exploiting the flexibility of nonparametric regression techniques (Hastie and Tibshirani 1990). Fully non-parametric proportional hazards models (O'Sullivan (1988) and Hastie and Tibshirani (1990)), while attractively flexible, usually suffer from heavy computational load and lack of formal inferential procedures. Gray (1994) used the concept of pseudo-smoothers, with emphasis on penalized B-splines, to develop formal inference for proportional hazards models. Penalized B-splines provide an elegant compromise between regression splines and smoothing splines.

Another issue in the analysis of time-to-event data is the modeling of multiple outcomes. This problem has received considerable attention in the statistical literature. For example, Wei, Lin and Weissfeld (1989) propose the use of marginal modeling. However, most available methods have not been extended to include flexible and possibly nonlinear effects of prognostic factors. On the other hand, many researchers have demonstrated that important prognostic factors (e.g. BMI) have a markedly non-linear effect on breast cancer survival and/or prognosis (Gray, 1994). These methods, however, are limited to single outcomes and do not lend themselves to simultaneous inference of several time-to-event outcomes.

In this article, we extend the marginal models of Wei, Lin and Weissfeld (1989) to allow modeling flexibility via the use of penalized B-splines in the style of Gray (1994). See also Hastie (1996) for a detailed discussion on a more general class of pseudo-smoothers. The remainder of the paper is organized as follows. In §2, we introduce the spline based proportional hazards model that fits a separate marginal model for each of several time-to-event outcomes. In §3, we discuss theoretical and computational details of the proposed

simultaneous inferential procedures. In §4, we present results from an extensive simulations study on the empirical size of the proposed tests in small sample settings. In §5, we present results from a detailed analysis of the BCPT data. The last section discusses the main findings of the paper and various modeling and model checking issues (including diagnostics measures) that extend the additive model to allow for testing the proportionality of hazards and multi-dimensional modeling.

2 The model

To model marginal distributions of multivariate time-to-event data, let us consider a flexible proportional hazards model for each of the G failure types. For the g^{th} type of failure of the i^{th}, $i = 1, ..., n$, subject, the model can be written as

$$\lambda_{gi}(t) = \lambda_{g0}(t) exp\{\sum_j f_{jg}(Z_{jgi})\}\ ,\ \ t \geq 0\ , \tag{1}$$

where $\lambda_{g0}(t)$ is an unspecified baseline hazard function and f_{jg}, $j = 1, ..., p$, denotes the unspecified smooth functions. In the usual setup (Cox, 1972), one observes data of the form $(X_{gi}, Z_{gi}, \Delta_{gi})$, where $X_{gi} = min(\tilde{X}_{gi}, C_{gi})$, C_{gi} is the censoring time, $Z_{gi}(t) = (Z_{1gi}(t), ..., Z_{pgi}(t))^T$ and $\Delta_{gi} = 1$ if $X_{gi} = \tilde{X}_{gi}$ and 0 otherwise.

Model (1) is fully non-parametric and quite general. Note also that the fully linear model of Wei, Lin and Weissfeld (1989) forms a special case of (1) where $f_{jg}(Z_{jgi}) = \beta_{jg} Z_{jgi}$. For this fully linear model, the partial likelihood is given as

$$PL_g(\beta) = \Pi_{i=1}^n \Big(\frac{exp\{\boldsymbol{\beta}_{g(T)} Z_{gi}(X_{gi})\}}{\sum_{l \in \mathcal{R}_g(X_{gi})} exp\{\boldsymbol{\beta}_{g(T)} Z_{gl}(X_{gl})\}} \Big)^{\Delta_{gi}}\ , \tag{2}$$

where $\boldsymbol{\beta}_g = (\boldsymbol{\beta}_{1g}, ..., \boldsymbol{\beta}_{pg})^T$ and $\mathcal{R}_g(t) = \{l : X_{gl} \geq t\}$ denotes the set of subjects at risk just prior to time t with respect to the g^{th} type of failure. The solution to $\partial log PL_g(\boldsymbol{\beta}_g)/\partial \boldsymbol{\beta}_g = 0$,

$\hat{\boldsymbol{\beta}}_g$, can be shown to be a consistent estimator of $\boldsymbol{\beta}_g$ provided that the fully linear model is correctly specified (Anderson and Gill, 1982).

In practical applications, the effects of most covariates are known to have some parametric form, while some of them are best modeled via non-parametric smoothers. For simplicity of discussion, we discuss most details for a model with p parametric and one additional non-parametric term. We first let

$$\lambda_{gi}(t) = \lambda_{g0}(t) exp\{\sum_j \beta_{jg} Z_{jgi} + f_g(h_{gi})\} \ , \ \ t \geq 0 \ , \tag{3}$$

where $j = 1, ..., p$. We propose to estimate $f_g(h_{gi})$ using the penalized regression spline approach, *i.e.*,

$$f_g(h_g) = \gamma_{1g} h_g + \sum_{q=2}^{m+3} \gamma_{qg} B_{qg}(h_g) \ . \tag{4}$$

Note that, we have dropped the constant term since it is accounted for by the baseline hazard, and only (m+2) of the B-spline basis functions are used for identifiability (De Boor, 1974). Following Gray (1994), let $\boldsymbol{\gamma}_g = (\gamma_{g2}, ..., \gamma_{g(m+3)})$ and $\boldsymbol{\eta}_g = (\gamma_{1g}, \boldsymbol{\gamma}_g)$. Then, a penalized partial likelihood that includes a penalty function to allow for smoother alternatives would be defined as

$$PL_g^p(\boldsymbol{\beta}_g, \boldsymbol{\eta}_g) = PL_g(\boldsymbol{\beta}_g, \boldsymbol{\eta}_g) - 1/2\lambda \int [f_g''(u)]^2 du \ . \tag{5}$$

Recognizing that the penalty function given above is quadratic in the parameter vector $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, ..., \gamma_{m+3})$, one could rewrite (5) as

$$PL_g^p(\boldsymbol{\beta}_g, \boldsymbol{\eta}_g) = PL_g(\boldsymbol{\beta}_g, \boldsymbol{\eta}_g) - 1/2\lambda_g \boldsymbol{\eta}_g^T \mathbf{K}_g \boldsymbol{\eta}_g \ . \tag{6}$$

where $\mathbf{K}$ is a positive definite matrix that is a function of the covariate h_g. Note that $\mathbf{K}$ is an $(m+3) \times (m+3)$ matrix with the first row and column as zeros, since the linear function passes unpenalized.

The hypotheses of interest with respect to the smooth function are then $\boldsymbol{\gamma}_g = \mathbf{0}$ and $\boldsymbol{\eta}_g = \mathbf{0}$, representing the hypotheses of "no effect" and "linear effect" respectively.

A model that is more focused towards testing proportionality of hazards via the use of time-varying coefficients could be considered as follows:

$$\lambda_{gi}(t) = \lambda_{g0}(t) exp\{\sum_j \beta_{jg} Z_{jgi} + \phi_g(t) h_{gi}\} \ , \ \ t \geq 0 \ . \tag{7}$$

It is straightforward to extend either of the above two models to allow for multiple, say M, non-parametric terms. In this case, $\boldsymbol{\eta}_g$ would be a bigger vector that augments contributions from the basis functions of the M terms. Here, $\boldsymbol{\eta}_g = (\boldsymbol{\eta}_{g1} : ... : \boldsymbol{\eta}_{gM})$ would be of dimension $M(m+3) \times 1$ and the penalty term would be the sum of the M penalty functions where each non-parametric term has its own smoothing parameter, and penalty matrix. One could then test for the "overall" effect or "linearity" of the individual non-parametric terms or for a combination of them.

3 Inference

While making inference on each of the margins is important, this could be done easily by using developments in Gray (1994). Our interest here is mainly in being able to conduct simultaneous inference on several time-to-event outcomes in models that have non-parametric smooth terms. Once the marginal distributions are modeled, then the methods described in Wei, Lin and Weissfeld (1989) can be extended to test for trends across parameter estimates and to combine estimates across margins to test for covariate effects of interest.

Let us consider the case where we have p parametric terms and one additional non-parametric term as given by (3). Then, for outcome g, the unpenalized part of equation (6),

suppressing the dependence of the regression parameters on X_{gi}, can be written as

$$PL_g(\boldsymbol{\beta}_g, \boldsymbol{\eta}_g) = \Pi_{i=1}^{n} \Big(\frac{exp\{\sum_{j=1}^{p} Z_{gj}\beta_{gj} + h_g\gamma_1 + \sum_{q=2}^{m+3} B_{qg}(h_g)\gamma_{qg}\}}{\sum_{s \in \mathcal{R}_g(X_{gi})} exp\{\sum_{j=1}^{p} Z_{gj}\beta_{gj} + h_g\gamma_1 + \sum_{q=2}^{m+3} B_{qg}(h_g)\gamma_{qg}\}} \Big)^{\Delta_{gi}} , \quad (8)$$

where all components are as defined in §2, for the g^{th} type of failure. Let $\boldsymbol{\psi}_g = (\boldsymbol{\beta}_g, \boldsymbol{\eta}_g)$ and $P_g = (Z_{1g} : ... : Z_{pg} : h_g : B_{2g}(h_g) : ... : B_{m+3,g}(h_g))$ with P_{gr} denoting the r^{th} column vector, $r = 1, ..., (m+p+3)$. Letting $\hat{A}_g$ be the unpenalized information matrix for the g^{th} outcome as a function of $\boldsymbol{\psi}$, it can be shown that

$$\sqrt{n}(\hat{\boldsymbol{\psi}}_g - \boldsymbol{\psi}_{g(T)}) = n(A_g + \lambda_n \tilde{K})^{-1} n^{-1/2} U_g(\boldsymbol{\psi}_{g(T)}) + o_p(1)$$

where $U_g(\boldsymbol{\psi}_{g(T)})$ is the score vector and $\boldsymbol{\psi}_{g(T)}$ is the vector of true parameter values for the g^{th} outcome (Gray, 1994) and $\tilde{K}$ is the expanded penalty matrix that augments rows and columns of zeros to $\mathbf{K}$ to account for the unpenalized terms in the model. Then, it follows from the asymptotic normality of $U_g(\boldsymbol{\psi}_{g(T)})$ that $\sqrt{n}(\hat{\boldsymbol{\psi}}_g - \boldsymbol{\psi}_{g(T)})$ is asymptotically normal with mean $\mathbf{0}$ and variance given as the limit of nV_g where

$$V_g = (A_g + \lambda_n \tilde{K})^{-1} A_g (A_g + \lambda_n \tilde{K})^{-1} , \quad (9)$$

To develop the simultaneous inferential procedures for several outcomes, we first note that the $\boldsymbol{\psi}_g$'s across the G multiple outcomes are generally correlated. Then, analogous to developments in Wei, Lin and Weissfeld (1989), the asymptotic covariance matrix between $\sqrt{n}(\hat{\boldsymbol{\psi}}_g - \boldsymbol{\psi}_g)$ and $\sqrt{n}(\hat{\boldsymbol{\psi}}_v - \boldsymbol{\psi}_v)$ can be consistently estimated by

$$\hat{D}_{gv}(\hat{\boldsymbol{\psi}}_g, \hat{\boldsymbol{\psi}}_v) = \hat{V}_g(\hat{\boldsymbol{\psi}}_g) \hat{C}_{gv}(\hat{\boldsymbol{\psi}}_g, \hat{\boldsymbol{\psi}}_v) \hat{V}_v(\hat{\boldsymbol{\psi}}_v) , \quad (10)$$

where $\hat{C}_{gv}(\hat{\boldsymbol{\psi}}_g, \hat{\boldsymbol{\psi}}_v) = n^{-1} \sum_{i=1}^{n} W_{gi}(\hat{\boldsymbol{\psi}}_g) W_{vi}(\hat{\boldsymbol{\psi}}_v)^T$, and W_{gi} and W_{vi} are defined in terms of the unpenalized score contributions as discussed in §4.1. below. Based on these results from §4.1, the covariance matrix of $(\hat{\boldsymbol{\psi}}_1, ..., \hat{\boldsymbol{\psi}}_G)$ can be consistently estimated by

$$\hat{Q} = n^{-1} \begin{pmatrix} \hat{D}_{11}(\hat{\psi}_1, \hat{\psi}_1) & \dots & \hat{D}_{1G}(\hat{\psi}_1, \hat{\psi}_G) \\ \vdots & \ddots & \vdots \\ \hat{D}_{G1}(\hat{\psi}_G, \hat{\psi}_1) & \dots & \hat{D}_{GG}(\hat{\psi}_G, \hat{\psi}_G) \end{pmatrix} . \tag{11}$$

3.1 Calculation of W_g

The robust variance estimator introduced by Wei, Lin and Weissfeld (1989) for inference across margins uses a plug-in estimator for covariances between the scores of the g^{th} and v^{th} margins.

For the g^{th} type of failure, let

$$N_{gi}(t) = I(X_{gi} \leq t, \Delta_{gi} = 1) \ ,$$

$$Y_{gi}(t) = I(X_{gi} \geq t)$$

and

$$M_{gi}(t) = N_{gi}(t) - \int_0^t Y_{gi}(u)\lambda_{gi}(u)du \ ,$$

where $I(.)$ denotes the indicator function. Then, it is straightforward to show that the penalized score function has the form

$$U_g^{(p)}(\psi_g) = U_g(\psi_g) - \lambda_g \tilde{K}_g \psi_g$$

where

$$U_g(\psi_g) = \sum_{i=1}^{n} \int_0^t P_{gi}(u) dM_{gi}(u)$$

$$- \int_0^t \frac{\sum_{i=1}^n Y_{gi}(u) P_{gi}(u) exp\{\psi_g^T P_{gi}(u)\}}{\sum_{i=1}^n Y_{gi}(u) exp\{\psi_g^T P_{gi}(u)\}} d\tilde{M}_g(u) \tag{12}$$

where $\bar{M}_g(u) = \sum_{i=1}^n M_{gi}(u)$. Based on arguments that are parallel to those in Wei, Lin and Weissfeld (1989), the asymptotic covariance matrix between $\sqrt{n}(\hat{\psi}_g - \psi_g)$ and $\sqrt{n}(\hat{\psi}_v - \psi_v)$ is given by

$$\hat{D}_{gv}(\hat{\psi}_g, \hat{\psi}_v) = \hat{V}_g(\hat{\psi}_g) E\{w_{g1}(\hat{\psi}_g) w_{v1}(\hat{\psi}_v)^T\} \hat{V}_v(\hat{\psi}_v) \ ,$$

where

$$w_{gj} = \int_0^i nf\{P_{gj}(t) - s_g^{(1)}(\psi_g; t)/s_g^{(0)}(\psi_g; t)\} dM_{gj}(t) \ ,$$

$$s_g^{(1)}(\psi_g; t) = E[Y_{gi}(t) P_{gi}(t) \exp\{\psi_g^T P_{gi}(t)\}] \ ,$$

and

$$s_g^{(0)}(\psi_g; t) = E[Y_{gi}(t) \exp\{\psi_g^T P_{gi}(t)\}] \ .$$

We then use a plug in estimate for $E\{w_{g1}(\hat{\psi}_g) w_{v1}(\hat{\psi}_v)^T\}$ which takes the form of $\hat{C}$ as in (10). This estimator is the same as the estimator proposed in Wei, Lin and Weissfeld (1989), since the penalty converges to zero under the null hypothesis. For this reason, the penalty term is dropped in the plug in estimate for $E\{w_{g1}(\hat{\psi}_g) w_{v1}(\hat{\psi}_v)^T\}$. We define,

$$W_{gi}(\psi_g) = \Delta_{gi} \Big\{ P_{gi}(X_{gi}) - \frac{S_g^{(1)}(\psi_g; X_{gi})}{S_g^{(0)}(\psi_g; X_{gi})} \Big\} - \sum_{l=1}^n \frac{\Delta_{gl} Y_{gi}(X_{gl}) exp\{\psi_g^T P_{gi}(X_{gl})\}}{n S_g^{(0)}(\psi_g; X_{gl})}$$

$$\times \Big\{ P_{gi}(X_{gl}) - \frac{S_g^{(1)}(\psi_g; X_{gl})}{S_g^{(0)}(\psi_g; X_{gl})} \Big\} \ , \tag{13}$$

$$S_g^{(1)}(\psi; t) = n^{-1} \sum_{i=1}^n Y_{gi}(t) P_{gi}(t) exp\{\psi_g^T P_{gi}(t)\} \ ,$$

and

$$S_g^{(0)}(\psi; t) = n^{-1} \sum_{i=1}^n Y_{gi}(t) exp\{\psi_g^T P_{gi}(t)\} \ .$$

The above asymptotic results are based on the approach used in Wei, Lin and Weissfeld (1989). Note that $\hat{Q}$ is constructed as a function of the information matrix, the penalty

matrix, the smoothing parameter and the individual elements of the unpenalized score vector, that is, a separate term is computed for each of the n observations. Note that, for the above approximation, the penalized versions of the likelihood and the score functions are used to compute the information matrix while the unpenalized score vector is used in the plug in estimator for the computation of W as given in (13). Note also that the penalty matrix $\tilde{K}_g$ contributes to the penalized score and information matrix only for the last $(m+2)$ components of ψ_g. Inferential procedures for the first p parametric terms are directly analogous to those outlined in Wei, Lin and Weissfeld (1989).

3.2 Testing statistical hypotheses

For the non-parametric term, one could conduct simultaneous inference on the "overall" effect and/or "linearity" of h across failure types. Let $\hat{\gamma}_g$ denote the components of $\hat{\psi}_g$ that correspond to the relevant components of the non-parametric term h_g. Let also $\hat{\Gamma}$ denote the relevant sub-matrix of $\hat{Q}$ corresponding to $\hat{\gamma} = (\hat{\gamma}_1, ..., \hat{\gamma}_G)$. Then, one could use the quadratic form $(\hat{\gamma}_1, ..., \hat{\gamma}_G)\hat{\Gamma}^{-1}(\hat{\gamma}_1, ..., \hat{\gamma}_G)^T$ to conduct a joint test on the null hypotheses given by $H_0 : \gamma_g = \mathbf{0}$, $g = 1, ..., G$. Note that the tests for "overall" significance or "linearity" are done in the above setup by choosing the last $(m+3)$ and $(m+2)$ elements of ψ_g respectively. A testing procedure that is more in the spirit of Gray (1994) uses $(A_g + \lambda_n \tilde{K}_g)^{-1}$ and $(A_v + \lambda_n \tilde{K}_v)^{-1}$ in (11) instead of V_g and V_v respectively. Under the null hypothesis, the modified Wald test statistic would then have an asymptotic distribution of

$$\sum_{g=1}^{G} \sum_{j} \lambda_{gj} \phi_j^2$$

where the ϕ_j are independent standard normal random variables, and the λ_{gj}'s are the eigenvalues of the matrix $\lim A_{\gamma\gamma|\psi}(A_{\gamma\gamma|\psi} + \lambda\tilde{K})^{-1}$, for the gth outcome. The arguments that lead to this form are given in Gray (1994) for a single outcome. The extensions to

multiple margins are straightforward. Note that the use of penalized B-splines, as opposed to fully nonparametric smoothers such as smoothing splines, makes the computation of the λ_{gj}'s possible.

A linear contrast could be constructed to test a group of parameters (e.g. all parameters to a spline term on each margin) across outcomes. For example, one could test the hypothesis that $\gamma_1 = ... = \gamma_G = \gamma$. One could then estimate the common γ by using a linear combination of the γ_g's in a way that takes the appropriate variances-covariance matrix into account. Unlike the tests discussed in Wei, Lin and Weissfeld (1989), where one is concerned with a single parameter from each margin, spline terms usually involve multiple parameters and the multicollinearity among them should be taken into account in taking the linear combinations via the off-diagonal covariance terms. Trends in regression effects across margins could also be examined along the lines of Wei, Lin and Weissfeld (1989) via sequential multiple testing procedures as in Wei and Stram (1988).

3.3 Choice of smoothing parameters, degrees of freedom, and placement of knots

In the above setup, we assume that the amount of smoothing (*i.e.*, the value of the smoothing parameter) is fixed by the analyst via prior knowledge or through a grid search. It is also possible that one could develop automatic procedures for selecting the smoothing parameters by using criteria such as cross validation. While this could lead to optimal estimation of the functional forms, its implications for hypothesis testing are not obvious. Operationally, one specifies the degrees of freedom per a non-parametric term and the corresponding value of smoothing parameter is then calculated. As a general operating guide, we use a relatively small number of degrees of freedom (Gray, 1994). The number of the knots that determine the B-spline basis functions are generally set to be at least twice the number of the degrees

of freedom in order to avoid wild fluctuation in the smooth function estimates, and are usually set to be between 10 and 15, per outcome. We will discuss the potential effects of various choices of the number of knots in our simulation studies. In this paper, we follow Gray (1994) in putting the knots at locations that yield approximately equal numbers of failure observations between knots. The calculation of degrees of freedom is analogous to that given in Gray (1994) and Wei, Lin and Weissfeld (1989). For example, to test whether all parameters in a spline model are equivalent across G outcomes, we use $\sum_{g=1}^{G} df_g$, where

$$df_g = \text{trace}\{\lim A^{(g)}_{\gamma\gamma|\psi}(A^{(g)}_{\gamma\gamma|\psi} + \lambda_g \tilde{K}_g)^{-1}\} \ .$$

4 Simulation Study

Extensive simulation studies were conducted to examine the performances of the proposed procedures for conducting simultaneous inference on several time-to-event outcomes. We focused on the bivariate case, where two time-to-event outcomes are considered under various levels of dependence. To generate data, the family of bivariate exponential distributions of Gumbel (1960) was used. Consider two marginal distributions, say F_1 and F_2, from the univariate exponential with hazard rates given by $\exp(\beta_1 Z)$ and $\exp(\beta_2 Z)$, respectively. Then, the distribution function of the bivariate exponential distribution is of the form

$$F(x_1, x_2) = F_1(x_1)F_2(x_2)[1 + \theta\{1 - F_1(x_1)\}\{1 - F_2(x_2)\}] \ .$$

The quantity $\theta/4$ measures the correlation between the two event times, where $-1 \leq \theta \leq 1$. In the above models, Z denotes any vector of covariates that may include binary indicators, or covariate effects that assume various functional forms.

In the simulations that test for overall significance, we set the covariate values in the two margins to be equal. Censoring indicators were generated independently using uniform

distributions gauged to depict various percentages of censoring (30%, 50%). Empirical sizes of the spline based tests, based on 2000 runs were examined under various specifications of sample sizes ($n = 200, 300, 400$), degrees of freedom ($df = 3, 5$), number of knots (10,15,20) and levels of dependence between the margins ($\alpha = 0.5, 1.0$). Note that the degree of correlation between the two outcomes is given by $\alpha/4$ and $\alpha = 1$ the maximum correlation allowed by the bivariate model of Gumbel (1960).

Table 1 gives results from simulation with low levels of dependence ($\alpha = 0.5$) between the outcomes. The results indicate that the empirical size is reasonably close to the corresponding nominal values only when the sample size is at least 200 per margin. Based on these simulation results and similar observations in Gray (1994), it would be advisable to use a smoother that has relatively small number of degrees of freedom, with number of knots not exceeding 15 for most practical applications.

(Table 1 around here)

Table 2 gives results from the simulation with high levels of dependence ($\alpha = 1.0$) between the outcomes. Here, due to the added level of dependence between the margins, the empirical sizes for $n = 200$ was still unacceptably high (results not shown). But, the empirical sizes for $n = 300, 400$ give reasonable results.

(Table 2 around here)

5 Example: The NSABP-BCPT Data

As an illustration of the proposed methods, we present results from a detailed analysis of data from the Breast Cancer Prevention Trial, hereafter refered to as BCPT, (Fisher et al, 1996). The BCPT was initiated in 1992 enrolling 13388 women that were at increased risk

for breast cancer due to their relatively old age ($\geq$60 years of age), relatively high 5-year predicted risk for breast cancer (a risk of at least 1.66% for those 35-59 years of age) and history of lobular carcinoma *in situ*. Subjects were then randomly assigned to placebo or treatment groups (6707 subjects into a placebo group and 6681 subjects receiving 20mg/day of tamoxifen for up to 5 years). The main aim was to examine the effectiveness of tamoxifen in preventing the possible occurences of invasive breast cancer in high-risk women. Data was also collected on other outcomes (some of them unwanted adverse side effects) such as invasive endometrial cancer, ischemic heart disease, transient ischemic attack, deep vein thrombosis and pulmonary embolism.

Analysis of data from the BCPT has shown (Fisher et al, 1998) that there was a 49% reduction in the risk of invasive breast cancer in those high risk women that received tamoxifen treatment (of up to five years) compared to those that received placebo. But, the benefits of tamoxifen were tempered by adverse side effects that significantly increased the risk of endometrial cancer, deep vein thrombosis, pulomanry embolism and some other cardiac effects. In fact, the issue of whether the benefits of tamoxifen outweighs the potential risk was controversial enough that the NCI sponsored a workshop on the subject in July, 1998, leading a risk-benefit analysis as reported in Gail et al. (1999).

The results indicate that age and baseline predicted risks for breast cancer play a significant role in determining whether the benefits of tamoxifen outweigh the associated risks. In this paper, we use the new developed techniques to simultaneously analyze several outcomes in a way that allows for risks that may not be constant across factors such as age. We focus on the invasive breast cancer (IBC), ischemic heart disease (IHD) and endometrial cancer (ENDO) as our outcomes of interest. The primary covariates of interest were treatment (TRT, placebo vs. tamoxifen), age at time of entry (AGE, in years), 5 year breast cancer risk at time of entry (based on a multivariate logistic model of Gail et al. (1989)) (PR5YR),

lobular carcinoma in situ (LCIS) and atypical hyperplasia of the breast (ATYPH, history at entry). The two continuous covariates that could be modeled using the spline approach, in order to examine non-linearity in their effects, were age and the five-year breast cancer probability from the Gail model.

The results from the marginal models on each of the three outcomes are given in Table 3 and the corresponding smooth function estimates for AGE and PR5YR are given in Figures 1-3. The results from the marginal models indicate that use of tamoxifen is associated with reduced risk of invasive of breast cancer ($p < 0.01$), but it was also associated with significantly increased risk of endometrial cancer. The increased risk in ischemic heart disease appeared to be marginal and not statistically significant. Age of the subjects appeared to be positively associated only with ischemic heart disease, but this association appeared to be linear (Figure 2). On the other hand, the 5yr probability of breast cancer (as estimated form Gail model) was non-linearly associated with onset of invasive breast cancer. Here, the estimated curve (Figure 1) indicates an initial rise in risk up to 6-7 units for the risk score with a decline in risk starting at about 10 units. The test for non-linearity was marginally significant indicating that a simple linear term may not suffice to control for this variable.

(Table 3 around here)

(Figures 1-3 around here)

The results from two bivariate models that simultineously model invasive breast cancer with ischemic heart disease and endometrial cancer are given in Table 4. The results indicate that the benefits of tamoxifen as a preventive agent significantly outweighs the side effect of increased risk in ischemic heart disease. On the other hand, the significant increased risk in endometrial cancer that is associated with the use of tamoxifen warrants a closer look since it appears to wash out its benefit of reducing the risk of breast cancer. However, these results should be interpreted cautiously due to the small number of events in the data set. The

results also indicate a strong linear effect of age in the bivariate model for invasive breast cancer and ischemic heart disease. Additionally, PR5YR appears to have a strong non-linear effect in both bivariate models, indicating that it should be modeled as a non-linear term.

(Table 4 around here)

6 Discussion

The methods proposed here have the advantage of being able to estimate a relatively realistic functional form for the covariate effects of interest, while enabling formal inference on the overall significance or adequacy of a certain parametric form (e.g. linearity) across several time-to-event outcomes. This is made possible through the use of penalized B-splines that are known to offer an attaractive compromise between fully non-parametric regression smoothers such as smoothing splines and flexible, but inherently parametric, techniques such as regression splines (Hastie and Tibshirani (1990), Gray (1994)).

In this paper, we have introduced a method for conducting simultaneous inference across several outcomes by extending the methods of Gray (1994) and Wei, Lin and Weissfeld (1989). The results from the analysis of the breast cancer data demonstrate its immediate usefulness in health related research. The simulated studies demonstrate that the asymptotic inferential procedures are reliable in finite sample settings and also provide rough guidelines on how to select realistic values for the degrees of freedom (hence smoothing parameters) and number and location of knots.

There are many open areas of research that would extend the methods in this paper, some of which are currently active areas of reasearch for our group. Some of the most important areas of research include dealing with proportionalty of hazards, diagnostic measures in the

multivariate setting, testing for trends in some parametric but monotonic subclass of the general spline approach (linearity has been explored here) and a more in depth examination of the issue of proportionality of hazards. A more general class of models that is based on the notion of pseudosplines as in Hastie (1996) is currently being developed by our group and results will be reported elsewhere. In this class of models, examination of adequacy of increasingly complex forms of polynomials would be natural due to the general structure of orthogonal-polynomial based pseudosplines, as opposed to the penalized B-splines discussed in this paper.

REFERENCES

Andersen, P.K. and Gill, R. D. (1982), "Cox's regression model for counting processes: A large sample study", The Annals of Statistics, 10, 1100-1120.

Cox, D.R. (1972), "Regression models and life tables (with discussion)", J. R. Statist. Soc. B 34, 187-220.

DeBoor, C. (1974), A Practical Guide to Splines. New York: Springer-Verlag.

Fisher, B., Dignam, J., Bryant, J., et al. (1996), "Five versus more than five years of tamoxifen therapy for breast cancer patients with negative lymph nodes and estrogen receptor-positive tumors", J. Natl. Cancer Inst. 88, 1529-1542.

Fisher, B.,Costantino, J.P., Wickerham, D.L. et al. (1998), "Tamoxifen for prevention of breast cancer: Report of the National Surgical Adjuvant Breast and Bowel Project P-1 Study", J. Natl. Cancer Inst. 90, 1371-1388.

Gail, M.H., Brinton, L.A., Bryan, D.P. et al. (1989), "Projecting individualized probabilities of developing breast cancer for white females who are being examined annually", J. Natl. Cancer Inst. 81, 1879-1886.

Gail, M.H., Costantino, J.P., Bryant, J. et al. (1999), "Weighing the risks and benefits of tamoxifen treatment for preventing breast cancer",

J. Natl. Cancer Inst. 91, 1829-1845.

Gumbel, E.J. (1960), "Bivariate exponential distributions", JASA 55, 698-707.

Gray, R. J. (1994), "Spline-Based test in survival analysis", Biometrics 50, 640-652.

Hastie, T. J. (1996). "Pseudosplines", J. R. Statist. Soc. B 58, 379-396.

Hastie, T. J. and Tibshirani, R. J. (1990a), Generalized Additive Models, London: Chapman and Hall.

Hastie, T. J. and Tibshirani, R. J. (1990b), "Exploring the nature of covariate effects in the proportional hazards model", Biometrics 46, 1005-1016.

O'Sullivan, F. (1988). "Nonparametric estimation of relative risk using splines and cross validation", SIAM J. Sci. and Stat. Comp. 9, 531-542.

Wei, L.J. and Stram, D.O. (1988), "Analysing repeated measurements with possibly missing observations by modelling marginal distributions", Statistics in Medicine 7, 139-148.

Wei, L. J., Lin, D. Y. and Weissfeld, L. (1989), "Regression analysis of multivariate incomplete failure time data by modeling marginal distributions", JASA 84, 1065-1073.

FIGURE LEGENDS:

1. Spline based estimates of the log hazard ratio for breast cancer as functions of age and five year probability of breast cancer

2. Spline based estimates of the log hazard ratio for ischemic heart disease as functions of age and five year probability of breast cancer

3. Spline based estimates of the log hazard ratio for endometrial cancer as functions of age and five year probability of breast cancer

Table 1: Empirical sizes of robust inference on marginally correlated ($\alpha = 0.5$) bivariate time-to-event outcomes

Censoring Prob.	Deg. of freedom	Number of knots	$n = 200$ Nominal level 0.01	0.05	0.10	$n = 300$ Nominal level 0.01	0.05	0.10
0.3	3	10	0.012	0.038	0.069	0.018	0.055	0.092
		15	0.022	0.070	0.121	0.029	0.079	0.130
		20	0.047	0.112	0.167	0.035	0.084	0.134
	5	10	0.030	0.068	0.114	0.022	0.071	0.121
		15	0.052	0.129	0.184	0.027	0.089	0.146
		20	0.103	0.200	0.270	0.051	0.137	0.206
0.5	3	10	0.013	0.051	0.096	0.013	0.051	0.089
		15	0.032	0.098	0.151	0.023	0.073	0.130
		20	0.074	0.163	0.238	0.041	0.120	0.185
	5	10	0.016	0.042	0.081	0.008	0.031	0.061
		15	0.029	0.080	0.124	0.015	0.046	0.083
		20	0.068	0.152	0.216	0.035	0.078	0.123

Table 2: Empirical sizes of robust inference on moderatelyy correlated ($\alpha = 1.0$) bivariate time-to-event outcomes

Censoring Prob.	Deg. of freedom	Number of knots	$n = 300$ Nominal level 0.01	0.05	0.10	$n = 400$ Nominal level 0.01	0.05	0.10
0.3	3	10	0.015	0.062	0.122	0.009	0.037	0.076
		15	0.033	0.092	0.156	0.012	0.051	0.086
		20	0.056	0.140	0.210	0.016	0.061	0.096
	5	10	0.028	0.085	0.144	0.012	0.045	0.081
		15	0.048	0.119	0.174	0.016	0.066	0.112
		20	0.078	0.166	0.237	0.024	0.073	0.131
0.5	3	10	0.022	0.085	0.172	0.004	0.025	0.051
		15	0.044	0.125	0.206	0.007	0.030	0.057
		20	0.066	0.171	0.263	0.010	0.049	0.086
	5	10	0.013	0.052	0.095	0.024	0.077	0.119
		15	0.023	0.078	0.136	0.029	0.086	0.156
		20	0.040	0.123	0.198	0.040	0.096	0.170

Table 3: Marginal Proportional Hazards Models on Breast Cancer, Ischemic Heart Disease and Endometrial Cancer

Outcome	Covariate	Estimate	Test Statistic	df	P-value
Invasive	TRT	-0.69	28.08	1	<0.01
Breast	LCIS	0.19	0.40	1	0.53
Cancer	AGE (overall)		2.89	4	0.61
	AGE (Linearity)		2.78	3	0.44
	PR5YR (overall)		17.26	4	<0.01
	PR5YR (Linearity)		6.94	3	0.05
Ischemic	TRT	0.13	0.59	1	0.44
Heart	LCIS	-0.95	2.00	1	0.16
Disease	AGE (overall)		73.3	3.99	<0.01
	AGE (Linearity)		3.54	3	0.30
	PR5YR (overall)		5.33	4	0.24
	PR5YR (Linearity)		2.96	3	0.40
Endometrial	TRT	0.88	8.23	1	<0.01
Cancer	LCIS	0.60	0.32	1	0.57
	AGE (overall)		4.32	3.99	0.36
	AGE (Linearity)		3.84	3	0.26
	PR5YR (overall)		5.19	4	0.25
	PR5YR (Linearity)		2.50	3	0.50

Table 4: Bivariate Proportional Hazards Models on Breast Cancer, Ischemic Heart Disease and Endometrial Cancer

Outcome	Covariate	Test Statistic	df	P-value
IBC	TRT	28.92	2	<0.01
and	LCIS	2.24	1.97	0.32
IHD	AGE (overall)	419.50	8	<0.01
	AGE (Linearity)	5.61	6	0.48
	PR5YR (overall)	24.80	8	<0.01
	PR5YR (Linearity)	10.93	6	0.07
IBC	TRT	36.57	2	<0.01
and	LCIS	0.44	2	0.62
ENDO	AGE (overall)	7.96	8	0.44
	AGE (Linearity)	7.29	6	0.27
	PR5YR (overall)	27.26	8	<0.01
	PR5YR (Linearity)	13.75	6	0.02

APPENDIX 2. COPY OF "ON THE USE OF PSEUDOSPLINES IN MODELING MULTIVARIATE SURVIVAL DATA: WITH APPLICATIONS TO THE NSABP-BCPT"

On the use of pseudosplines in modeling multivariate survival data: with applications to the NASBP-BCPT

Kiros Berhane and Lisa A. Weissfeld *

---

* Kiros Berhane is Assistant Professor, Department of Preventive Medicine, University of Southern California, 1540 Alcazar Street CHP220, Los Angeles, CA 90033. Lisa A. Weissfeld is Professor, Department of Biostatistics, University of Pittsburgh, 303 Parran Hall, Pittsburgh, PA 15261. This research was supported by an idea award from the U.S. Department of Defence (DAMD17-99-9356)

Abstract

Pseudosmoothers have been applied to time-to-event data, providing an extension of the proportional hazards model for a single outcome (Gray, 1994). We use this technique to extend the marginal models of Wei, Lin and Weissfeld (1989), by also allowing for a more flexible class of models than that allowed in Gray (1994) along the lines of Hastie (1996). This allows for greater flexibility in modeling the margins and makes formal development of inferential procedures possible. This method is illustrated with an example using data from the NSABP trial and small sample proporties on the size and power of the proposed tests are studied via simulated data.

KEY WORDS: Survival analysis; Smoothing; Ridge regression; Additive models; Splines.

1 INTRODUCTION

The proportional hazards model (Cox, 1972), a widely popular method of analyzing censored time-to-event data, has been recently extended to allow data-dependent, and possibly non-linear, covariate effects by exploiting the flexibility of nonparametric regression techniques (Hastie and Tibshirani 1990a). These extensions include the fully non-parametric methods of O'Sullivan (1988) and Hastie and Tibshirani (1990b). These models, while attractively flexible, usually suffer from heavy computational load and lack of formal inferential procedures. Gray (1994) used the concept of penalized B-splines to develop formal inference for proportional hazards models. Gray's model sets up an inherently parametric model by using B-spline basis functions (De Boor, 1974), but then penalizes them to allow for smoother alternatives. The end result is a model that is flexible enough to capture non-linearities in covariate effects, but also allows for formal inference due to its quasi-parametric structure.

To date, non-parametric regression models have not been developed for multivariate time-to-event outcomes. In this article, we extend the marginal models of Wei, Lin and Weissfeld (1989) to allow data-dependent estimation of possibly nonlinear effects of covariates. Our extensions are analogous to those of Gray (1994). But, we use the more general class of pseudosplines that has been proposed by Hastie (1996). Specifically, we would like to exploit the ordered complexity of orthogonal polynomial basis functions in

order to more closely examine trends in covariate effects. This also naturally lends itself to testing of the adequacy of polynomial functions as opposed to fully non-parametric modeling techniques. An imporatant subclass of this rich class of models is the one that has time-varying coeffcients of covariate effects, as it allows for a check on the proportionalty of hazards (Zucker and Karr, 1990).

The methods in this article are motivated by applications to data from the Breast Cancer Prevention Trial (BCPT, Fisher et al. 1998). The main purpose of this study was to examine the usefullness of the drug tamoxifen as an agent that could potentially prevent invasive breast cancer. But, the study also collected data on some of the drug's known adverse effects, such as ischemic heart disease, endometrial cancer and pulmonary embolism. We wanted to analyze the multiple time-to-event data that arose from this study in a way that allows for simultaneous inference across the various outcomes, but also allowing for nonlinear effects of important risk factors.

The remainder of the paper is organized as follows. In §2, we present the new flexible marginal models for multiple-to-event data. In §3, we present result from extensive simulation studies to study the small sample properties of the proposed inferential procedures. §4 summarizes the results from applications of the proposed methodology to data from the NSABP breast cancer prevention trial (BCPT). In §5, we summarize the main results and give details on future directions for research. The details on the theoreti-

cal development and asymptotic properties of the inferential procedures are given in the Appendix (?).

2 Proposed Model

2.1 Pseudo-smoothers

To fix ideas, we first consider linear smoothers in the univariate framework. Letting $(x_1, y_1), ..., (x_n, y_n)$ denote a set of n independent observations, a scatterplot smoother is said to be linear if, concentrating on the computations of the function only at the design points in $\mathbf{x} = (x_1, ..., x_n)$, it can be written as a linear map $S : R^n \rightarrow R^n$ defined by $\hat{\mathbf{y}} = S\mathbf{y}$, where $\mathbf{y} = (y_1, ..., y_n)$ is the response vector. Here S is referred to as a smoother matrix and is analogous to the hat matrix in linear regression. From this point onwards, our discussion focuses on the smoothing spline, even though the idea of pseudo-smoothers applies, in principle, to any linear smoother.

A cubic smoothing spline minimizes the penalized least squares criterion

$$\sum_{i=1}^{n}\{y_i - g(x_i)\}^2 + \alpha \int_{-\infty}^{+\infty} g''(z)^2 dz \ , \tag{1}$$

over a suitable Sobolev space W_2 of functions (Wahba 1990). An equivalent form of (1), that is based on the fitted vector $\mathbf{f} = (f_1, ..., f_n)$ and a penalty matrix $\mathbf{K}$, is

$$(\mathbf{y}-\mathbf{f})^T(\mathbf{y}-\mathbf{f})+\alpha\mathbf{f}^T\mathbf{K}\mathbf{f}\ . \tag{2}$$

The solution to (2), which is a natural cubic spline with knots at each distinct x_i, can be shown to be

$$\hat{\mathbf{f}}=(\mathbf{I}+\alpha\mathbf{K})^{-1}\mathbf{y}\ , \tag{3}$$

where $S=(\mathbf{I}+\alpha\mathbf{K})^{-1}$ is the smoother matrix.

Based on observations from the eigendecomposition of S, Hastie (1996) proposed a class of pseudo-smoothers that can be constructed from a given set of orthonormal basis functions (which should be ordered in complexity) and a penalty sequence.

Letting $\mathbf{p}(x)$ be a $M\times 1$ vector of orthonormal basis functions and $\theta_m, m=1,...,M$ the penalties, a pseudospline could be defined as a minimizer of

$$Q_\lambda=(\mathbf{y}-P\boldsymbol{\beta})^T(\mathbf{y}-P\boldsymbol{\beta})+\alpha\boldsymbol{\beta}^T D_\theta\boldsymbol{\beta}\ , \tag{4}$$

where P is the matrix of evaluations of $\mathbf{p}$ at the data, $D_\theta=diag(\theta_1,...,\theta_M)$ and α is a smoothing parameter. Then, the solution can be shown to be

$$\hat{\mathbf{f}}=P\boldsymbol{\beta}=P(I+\alpha D_\theta)^{-1}P^T\mathbf{y} \tag{5}$$

provided that the the bases are orthonormal with respect to the observed $\mathbf{x}$, leading to a ridge type regression setting. To approximate the smoother matrix of a smoothing spline, S, we supply a pseudobasis P in order to define a pseudo-eigendecomposition of S, *i.e.*,

$$\hat{S}(P) = PD_{\psi}P^T \ . \tag{6}$$

The pseudo-eigenvalues that form the $M \times M$ diagonal matrix $D_{\psi} = diag(\psi_m)$ can be obtained by $\psi_m = \mathbf{p}^T S\hat{\mathbf{p}}_m$, where $\hat{\mathbf{p}}_m$ is a result of smoothing $\mathbf{p}_m$ in O(n) computations. Hastie (1996) proposes a better approximation, $\hat{S}(P^*)$, obtained by considering the $M \times M$ eigendecomposition $P^TSP = VD_{\psi}^*V^T$ and defining $P^* = PV$.

To define a pseudosmoother, one needs to supply a set of basis functions and a penalty sequence. The rank (M) required to approximate S increases with the degrees of freedom ($tr(S)$). The use of penalized partial likelihood for a model that is defined via B-spline basis functions (Gray, 1994) is one example. There are many other choices for defining the pseudobasis functions (see Hastie 1996).

For weighted smoothers, the idea of approximation remains the same while the level of complications depends on the structure of the weight matrix. For a diagonal weight matrix with positive elements, the algorithm remains the same following premultiplication of the x and y values by square root of the respective weights. For iteratively weighted additive models (where the

weights change at each iteration), we use the above algorithm to approximate the penalty matrix K (instead of S itself). We will discuss the mechanism for doing this (and reasons for it) in the next section.

2.2 The model

To model marginal distributions of multivariate time-to-event data, let us consider a flexible proportional hazards model for each of the G failure types. For the g^{th} type of failure of the i^{th}, $i = 1, ..., n$, subject, the model can be written as

$$\lambda_{gi}(t) = \lambda_{g0}(t) exp\{\sum_j f_{jg}(Z_{jgi})\} \ , \ \ t \geq 0 \ , \tag{7}$$

where $\lambda_{g0}(t)$ is an unspecified baseline hazard function and f_{jg}, $j = 1, ..., p$, denoting unspecified smooth functions. In the usual setup (Cox, 1972), one observes data of the form $(X_{gi}, Z_{gi}, \Delta_{gi})$, where $X_{gi} = min(\tilde{X}_{gi}, C_{gi})$, C_{gi} is the censoring time, $Z_{gi}(t) = (Z_{1gi}(t), ..., Z_{pgi}(t))^T$ and $\Delta_{gi} = 1$ if $X_{gi} = \tilde{X}_{gi}$ and 0 otherwise. Then, the partial likelihood for the fully linear model, where $f_{jg}(Z_{jgi}) = \beta_{jg} Z_{jgi}$, is given as

$$PL_g(\beta) = \Pi_{i=1}^n \Big(\frac{exp\{\boldsymbol{\beta}_g^T Z_{gi}(X_{gi})\}}{\sum_{l \in \mathcal{R}_g(X_{gi})} exp\{\boldsymbol{\beta}_g^T Z_{gl}(X_{gl})\}} \Big)^{\Delta_{gi}} \ , \tag{8}$$

where $\boldsymbol{\beta}_g = (\boldsymbol{\beta}_{1g}, ..., \boldsymbol{\beta}_{pg})^T$ and $\mathcal{R}_g(t) = \{l : X_{gl} \geq t\}$ denotes the set of subjects at risk just prior to time t with respect to the g^{th} type of failure.

The solution to $\partial log PL_g(\boldsymbol{\beta}_g)/\partial \boldsymbol{\beta}_g = 0$, $\hat{\boldsymbol{\beta}}_g$, can be shown to be a consistent estimator of $\boldsymbol{\beta}_g$ provided that the fully linear model is correctly specified (REF).

Then, a penalized partial likelihood, letting $\eta_g = \sum_j f_{jg}(Z_{jgi})$, would be given as

$$j_g(\eta_g) = PL_g(\eta_g) - 1/2 \sum_{i=1}^{p} \alpha_{gi} \int f_g''(s)^2 ds \ . \tag{9}$$

In terms of the fitted vectors, this can be given as

$$j_g(\eta_g) = PL_g(\eta_g) - 1/2 \sum_{i=1}^{p} \alpha_{gi} \mathbf{f}_{gi}^T \mathbf{K}_{gi} \mathbf{f}_{gi} \ . \tag{10}$$

The Newton-Raphson algorithm for maximizing $j_g(\eta_g)$ over $\mathbf{f}_{g1}, ..., \mathbf{f}_{gp}$, letting $\mathbf{u}_g = dl/d\eta_g$ and $\mathbf{A}_g = -d^2 l/d\eta_g \eta_g^T$ denote the unpenalized score vector and information matrix respectively, in order to get updates of $\mathbf{f}_{1g}^{(0)}, ..., \mathbf{f}_{1g}^{(0)}$ to $\mathbf{f}_{1g}^{(1)}, ..., \mathbf{f}_{1g}^{(1)}$ is (suppressing the subscript g for now)

$$\begin{pmatrix} \mathbf{A}+\alpha_1\mathbf{K}_1 & \mathbf{A} & \dots & \mathbf{A} \\ \mathbf{A} & \mathbf{A}+\alpha_2\mathbf{K}_2 & \dots & \mathbf{A} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A} & \mathbf{A} & \dots & \mathbf{A}+\alpha_p\mathbf{K}_p \end{pmatrix} \begin{pmatrix} \mathbf{f}_1^{(1)} - \mathbf{f}_1^{(0)} \\ \mathbf{f}_2^{(1)} - \mathbf{f}_2^{(0)} \\ \vdots \\ \mathbf{f}_p^{(1)} - \mathbf{f}_p^{(0)} \end{pmatrix} = \begin{pmatrix} u - \alpha_1\mathbf{K}_1\mathbf{f}_1^{(0)} \\ u - \alpha_2\mathbf{K}_2\mathbf{f}_2^{(0)} \\ \vdots \\ u - \alpha_p\mathbf{K}_p\mathbf{f}_p^{(0)} \end{pmatrix} . \tag{11}$$

Here, $\mathbf{K}_j$ represents a quadratic penalty matrix for the j^{th} predictor. From (11), one gets the $np \times np$ system of estimating equations

$$\begin{pmatrix} \mathbf{I} & \mathbf{S}_1 & \mathbf{S}_1 & \dots & \mathbf{S}_1 \\ \mathbf{S}_2 & \mathbf{I} & \mathbf{S}_2 & \dots & \mathbf{S}_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{S}_p & \mathbf{S}_p & \mathbf{S}_p & \dots & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{f}_1^{(1)} \\ \mathbf{f}_2^{(1)} \\ \vdots \\ \mathbf{f}_p^{(1)} \end{pmatrix} = \begin{pmatrix} \mathbf{S}_1\mathbf{z}^{(0)} \\ \mathbf{S}_2\mathbf{z}^{(0)} \\ \vdots \\ \mathbf{S}_p\mathbf{z}^{(0)} \end{pmatrix} , \tag{12}$$

where $\mathbf{z}_g^{(0)} = \boldsymbol{\eta}_g^{(0)} + \mathbf{A}_g^{-1(0)}\mathbf{u}_g^{(0)}$ and $\mathbf{S}_{jg} = (\mathbf{A}_g + \alpha_{jg}\mathbf{K}_{jg})^{-1}\mathbf{A}_g$ for the g^{th} type of failure.

In this paper, we use the pseudospline technology of Hastie (1996) to approximate the fully non-parametric smoothing splines. This allows us to make the computation manageable. More importantly, it allows us to develop formal techniques for statistical inference.

The backfitting algorithm that follows from (11) requires an expensive operation ($O((n)^3)$ operations) in order to update each smooth function in (9). For large data sets with many predictor variables, this may not be practical. We propose to approximate each smoother matrix S_{jg}, $j = 1, ..., p$, by a rank M_{jg} approximation.

Following Hastie (1996), we choose to approximate the quadratic penalty matrices K_{jg} (instead of the smoother matrices S_{jg}'s) for each nonparametric term in (9). This approach has two main advantages. First, the approximation of the K_{jg}'s can be done in the unweighted setting to be followed by a weighted generalized ridge regression in order to accommodate the weights. Second, there is no need to make fresh approximations at each iterative step of the outer loop in the local scoring algorithm. From our discussions in §2.1 and details in Hastie (1996), it is easy to show that the steps that we outlined for approximating the S_{jg}'s are still applicable in approximating the K_{jg}'s. The resulting orthogonal basis matrices $\mathbf{P}_{jg}$'s for the S_{jg}'s are also the eigenvectors of the K_{jg}'s and the corresponding eigenvalues for K_{jg}'s could

be obtained by using $\boldsymbol{\psi}_{jg} = (1/(1+\alpha_{jg}\boldsymbol{\theta}_{jg}))$, where $\boldsymbol{\psi}_{jg}$ and $\boldsymbol{\theta}_{jg}$ are $M_{jg} \times 1$ vectors of penalties corresponding to S_{jg} and K_{jg} respectively.

Now, let us replace each of the penalty matrices, K_{jg}, in (11) by a rank M_{jg} approximation $\hat{K}_{jg} = P_{jg}D_{\theta_{jg}}P_{jg}^T$ and let $\mathbf{f}_{jg} = P_{jg}\boldsymbol{\beta}_{jg}$. Then (suppressing the subscript g),

$$\begin{pmatrix} \mathbf{A}+\alpha_1 P_1 D_{\theta_1} P_1^T & \mathbf{A} & \dots & \mathbf{A} \\ \mathbf{A} & \mathbf{A}+\alpha_2 P_2 D_{\theta_2} P_2^T & \dots & \mathbf{A} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A} & \mathbf{A} & \dots & \mathbf{A}+\alpha_p P_p D_{\theta_p} P_p^T \end{pmatrix} \begin{pmatrix} P_1\boldsymbol{\beta}_1^{(1)} - P_1\boldsymbol{\beta}_1^{(0)} \\ P_2\boldsymbol{\beta}_2^{(1)} - P_2\boldsymbol{\beta}_2^{(0)} \\ \vdots \\ P_p\boldsymbol{\beta}_p^{(1)} - P_p\boldsymbol{\beta}_p^{(0)} \end{pmatrix}$$

$$= \begin{pmatrix} u - \alpha_1 P_1 D_{\theta_1} P_1^T P_1 \boldsymbol{\beta} 1^{(0)} \\ u - \alpha_2 P_2 D_{\theta_2} P_2^T P_2 \boldsymbol{\beta}_2^{(0)} \\ \vdots \\ u - \alpha_p P_p D_{\theta_p} P_p^T P_p \boldsymbol{\beta}_p^{(0)} \end{pmatrix} . \tag{13}$$

Letting $P_g = (P_{1g} : P_{2g} : ... : P_{pg})$, $D_{\theta_g} = Bdiag(D_{\theta_{jg}})$ and $\boldsymbol{\beta}_g = (\boldsymbol{\beta}_{1g}^T : ... : \boldsymbol{\beta}_{pg}^T)$ for the g^{th} type of failure, it is easy to show that (13) is equivalent to

$$P_g\boldsymbol{\beta}_g^{(1)} = (\mathbf{A}_g + P_g D_{\theta_g} P_g^T)^{-1}\mathbf{A}_g\mathbf{z}_g^{(0)} ,$$

where $\mathbf{z}_g^{(0)} = \mathbf{A}_g^{-1(0)}\mathbf{u}_g^{(0)} + P_g\boldsymbol{\beta}_g^{(0)}$ and hence

$$\boldsymbol{\beta}_g^{(1)} = (P_g^T\mathbf{A}_g^{(0)}P_g + D_{\theta_g})^{-1}P_g^T\mathbf{A}_g^{(0)}\mathbf{z}_g^{(0)} . \tag{14}$$

For unweighted generalized ridge regression, a well known trick (Golub and Van Loan 1983) could reduce the problem to that of an ordinary least

squares. This is done by creating pseudo-design matrix and pseudo-response vectors of the form

$$P_g^a = \begin{pmatrix} P_g \\ D_{\theta_g}^{1/2} \end{pmatrix}$$

and

$$\mathbf{X}_g^* = \begin{pmatrix} \mathbf{X}_g \\ \mathbf{0} \end{pmatrix} .$$

This allows for efficient computation following a Q-R decomposition of P_g^a. See Hastie (1996, Appendix A.2) for details. When the weight matrix is diagonal, the same trick could be used after premultiplying P_g and $\mathbf{X}_g$ by square-root of the corresponding weights. For the fully weighted proportional hazards framework, however, this trick is less appealing since it requires the formation of matrix square roots of the weight matrix and new Q-R decompositions at each iteration. The use of matrix square roots for $\mathbf{A}$ is also bound to lead to ill-conditioning and numerical instability (Thisted 1988, §3.10.3).

Following Hastie and Tibshirani (1990), one could use a diagonal approximation of the weight matrix. This is shown to give the correct penalized maximum likelihood estimator and justified via the delta algorithm (Jorgensen, 1984). So, considerable saving in computation could be achieved by using the above outlined trick after proper pre-multiplications of the P_g's and the $\mathbf{X}_g$'s by the square root of the diagonal weights.

2.3 Inference

Once the marginal distributions are modeled, then the methods described in Wei, Lin and Weissfeld (1989) can be extended to test for trends across parameter estimates and to combine estimates across margins to test for covariate effects of interest. Specifically, based on equation (14) the partial likelihood can be written as

$$PL_g(\beta_g) = \Pi_{i=1}^{n}\Big(\frac{exp\{P_{g1}\boldsymbol{\beta}_{g1}(X_{gi}) + ... + P_{gp}\boldsymbol{\beta}_{gp}(X_{gi})\}}{\sum_{l\in\mathcal{R}_g(X_{gi})} exp\{P_{g1}\boldsymbol{\beta}_{g1}(X_{gl}) + ... + P_{gp}\boldsymbol{\beta}_{gp}(X_{gl})\}}\Big)^{\Delta_{gi}} \quad , (15)$$

where $\boldsymbol{\beta}_{gr} = (\beta_{gr1}, ..., \beta_{grm})^T$ and P_{gr} is the $m \times m$ orthogonal basis matrix, which is also a function of X, for the g^{th} type of failure and the r^{th} covariate. Define

$$W_{gir}(\hat{\boldsymbol{\beta}}_{gr}) = \Delta_{gi}\Big(\sum_{j=1}^{m} P_{gr(jr)}(X_{gi}) - \frac{S_g^{(1)}(\hat{\boldsymbol{\beta}}_{gr}; X_{gi})}{S_g^{(0)}(\hat{\boldsymbol{\beta}}_{gr}; X_{gi})}\Big) - \sum_{j=1}^{n} \frac{\Delta_{gj} Y_{gi}(X_{gj}) exp(\sum_{k=1}^{p} P_{gk}\boldsymbol{\beta}_{gk}(X_{gj})}{nS_g^{(0)}(\boldsymbol{\beta}_{gr}; X_{gj})}$$

$$\Big(\sum_{j=1}^{m} P_{gr(jr)}(X_{gj}) - \frac{S_g^{(1)}(\boldsymbol{\beta}_{gr}; X_{gj})}{S_g^{(0)}(\boldsymbol{\beta}_{gr}; X_{gj})}\Big) \quad , \qquad (16)$$

where

$$S_g^{(1)}(\boldsymbol{\beta}_r; t) = n^{-1}\sum_{i=1}^{n} Ygi(t)(\sum_{j=1}^{m} P_{gr(jr)})exp(\sum_{s=1}^{m} P_{gs}\boldsymbol{\beta}_{gs}(t)) \quad ,$$

$$S_g^{(0)}(\boldsymbol{\beta}_r; t) = n^{-1}\sum_{i=1}^{n} Ygi(t)exp(\sum_{s=1}^{m} P_{gs}\boldsymbol{\beta}_{gs}(t)) \quad ,$$

and $Y_{gi}(t) = I(X_{gi} \geq t)$. Then, the asymptotic covariance matrix between $\sqrt{(n)}(\hat{\beta}_{ui} - \beta_{ui})$ and $\sqrt{(n)}(\hat{\beta}_{vh} - \beta_{vh})$ can be consistently estimated by

$$\hat{D}_{uv}(\hat{\beta}_{ui}, \hat{\beta}_{vh}) = \hat{I}_u^{-1}(\hat{\beta}_{ui})\hat{\beta}_{uv}(\hat{\beta}_{ui}, \hat{\beta}_{vh})\hat{I}_v^{-1}(\hat{\beta}_{vh}) \ , \tag{17}$$

where $\hat{I}_v^{-1}(\hat{\beta}_{vi})$ is the inverse of the negative of the second derivative matrix of the partial likelihood based on (14) and $\hat{\beta}_{uv}(\hat{\beta}_{ui}, \hat{\beta}_{vh}) = n^{-1} \sum_{j=1}^{n} W_{uji}(\hat{\beta}_{ui}) W_{vjh}(\hat{\beta}_{vh})^T$, where W_{uji} and W_{vjh} are defined in (16). Thus, the covariance matrix of $(\hat{\beta}_1, ..., \hat{\beta}_p)$ can be consistently estimated by $\hat{Q} = n^{-1}[D_{11}]$.

REFERENCES

Cox, D.R. (1972), "Regression models and life tables (with discussion)", J. R. Statist. Soc. B 34, 187-220.

Golub, G. H. and Van Loan, C. F. (1983), Matrix Computations, Baltimore: Johns Hopkins University Press.

Gray, R. J. (1994), "Spline-Based test in survival analysis", Biometrics 50, 640-652.

Hastie, T. J. (1996). "Pseudosplines", J. R. Statist. Soc. B 58, 379-396.

Hastie, T. J. and Tibshirani, R. J. (1990a), Generalized Additive Models, London: Chapman and Hall.

Hastie, T. J. and Tibshirani, R. J. (1990b), "Exploring the nature of covariate effects in the proportional hazards model", Biometrics 46, 1005-1016.

Jorgensen, B (1984). "The delta algorithm and GLIM", Int. Stat. Rev. 52, 283-300.

O'Sullivan, F. (1988). "Nonparametric estimation of relative risk using splines and cross validation", SIAM J. Sci. and Stat. Comp. 9, 531-542.

Thisted, R. A. (1988), Elements of Statistical Computing, London: Chapman and Hall.

Wahba, G. (1990), Spline Functions for Observational Data, CBMS-NSF Regional

Conf. Series, SIAM. Philadelphia.

Wei, L. J., Lin, D. Y. and Weissfeld, L. (1989), "Rgeression analysis of multivariate

incomplete failure time data by modeling marginal distributions", JASA 84,

1065-1073.

APPENDIX 3. COPY OF "ESTIMATION OF THE SURVIVAL FUNCTION FOR GRAY'S PIECEWISE-CONSTANT TIME-VARYING COEFFICIENTS MODEL"

ESTIMATION OF THE SURVIVAL FUNCTION FOR GRAY'S PIECEWISE-CONSTANT TIME-VARYING COEFFICIENTS MODEL

ZDENEK VALENTA *†AND LISA WEISSFELD

Department of Biostatistics, University of Pittsburgh, 130 DeSoto Street, 303 Parran Hall,

Pittsburgh, PA 15261, U.S.A.

*Correspondence to: Zdenek Valenta, Department of Biostatistics, University of Pittsburgh, 130 DeSoto Street, 303 Parran Hall, Pittsburgh, PA 15261, U.S.A.

†E-mail: zdvst@imap.pitt.edu Phone: (412) 624-3023 Fax: (412) 624-2183

Contract/grant sponsor: NIDDK; contract/grant number: R01 HS09694-03

Contract/grant sponsor: Department of the Army; contract/grant number: DAMD17-99-1-9536

SUMMARY

Gray's extension of Cox's proportional hazards (PH) model for right-censored survival data allows for a departure from the PH assumption via introduction of time-varying regression coefficients (TVC). For this model estimation of the conditional hazard rate relies on the inclusion of penalized splines. Cubic penalized splines tend to be unstable in the right tail of the distribution and thus quadratic, linear and piecewise-constant penalized splines may be a favorable choice. We derive a survival function estimator for one important member of the class of TVC models - a piecewise-constant time-varying coefficients (PC-TVC) model. Using the first-order Taylor series approximation we also derive an estimate for the variance of the log- and log(-log)-transformed survival function, which in turn leads to estimated confidence limits on the corresponding scales of the survival function. Accuracy in estimating underlying survival times and survival quantiles is assessed for both Cox's and Gray's PC-TVC model using a simulation study featuring scenarios violating the PH assumption. Finally, an example of the estimated survival functions and the corresponding confidence limits derived from Cox's PH and Gray's PC-TVC model, respectively, is presented for a liver transplant data set.

1. INTRODUCTION

The Cox proportional hazards (PH) model has played a prominent role in both the statistical literature and for the analysis of right-censored survival data since its first introduction by

Cox [1] in 1972. It has been widely used for the analyses of biomedical data from both longitudinal studies and clinical trials, mainly due to its appealing mathematical simplicity, as well as its general availability through most statistical packages. While the Cox PH model is relatively simple to present, it relies on the assumption of proportionality which may not be met in all data sets. To address this issue, models that allow for non-proportionality of the conditional hazards through the introduction of penalized splines have been proposed. A family of models which can be used to model non-proportional data, the time-varying coefficient (TVC) models, have been considered by Gamerman and West [2], and Zucker and Karr [3]. A general treatment of the first order asymptotic analysis of the penalized likelihood is due to Cox and O'Sullivan [4]. Building on the work of Tsiatis [5], Andersen and Gill [6] and Gill [7], O'Sullivan [8] treated nonparametric estimation in the Cox model using an approach complementary to that of Zucker and Karr [3]. The methodology of Zucker and Karr was further developed by Gray [9, 10]. Time-varying coefficient models were also studied by Hastie and Tibshirani [11] and the use of regression splines in modeling the conditional hazard rate is discussed in Sleeper and Harrington [12] and Gray [9]. The use of time dependence in Cox's PH model was also investigated by Pettitt and Daud [13], Hess [14] and Verweij and van Houwelingen [15]. One of the more useful spline-based extensions of the Cox proportional hazards model is that proposed by Gray [9]. Gray's TVC extension of the Cox PH model employs products of the covariates of interest with the spline functions of time. This allows for a flexible approach to the modeling of covariate effects without necessarily adhering to the assumption of proportional hazards, which may not be satisfied. The most

appealing model within the framework of models proposed by Gray is the piecewise-constant TVC (Gray's PC-TVC) model since this model is similar to the original Cox PH model and retains much of the mathematical simplicity of the Cox model. The advantage of the PC-TVC models is their flexibility, since the proportional hazards assumption is only required for each of the time intervals between the successive knots (i.e. time points allowing for a change in the regression coefficients). Gray's PC-TVC model may therefore be viewed as a piecewise proportional hazards model for the conditional hazard rate. The estimated survival function is often of interest when fitting a survival model to data, since this serves as a useful summary of the estimated survival experience of a given population. Gray's work on TVC models has focused on estimation of the model coefficients, inference and residual analysis and to date, no estimator for the survival function has been presented. Andersen *et al.* [16] show that confidence limits for the survival function estimated from the Cox PH model are optimal when the estimates are based on a log-transformed or log(-log)-transformed scale for the survival curve. In this paper we present an estimator of the survival function under Gray's PC-TVC model. Estimation is based on the observation that between the successive knots, where the hazard regression coefficients are assumed to remain constant, the integration with respect to a differential of the cumulative hazard rate may proceed in a manner similar to that for the original Cox PH model. The estimated variance of the predicted survival function under Gray's PC-TVC model is derived for both the log- and log(-log)-transformed scale of the survival function and corresponding estimates of the confidence limits are presented.

2. ESTIMATED SURVIVAL FOR GRAY'S PC-TVC MODEL

Within the TVC family of models we assume that the hazard function can be modeled as follows:

$$d\Lambda\left(t|x\right) = d\Lambda_0\left(t\right) exp\left\{x'\beta\left(t\right)\right\}, \tag{1}$$

where $\Lambda(.)$ denotes the cumulative hazard function and $\Lambda_0(.)$ denotes the cumulative baseline hazard. Here $\beta'(t) = (\beta_1(t), \beta_2(t), \ldots, \beta_p(t))$, where $\beta_j\left(t\right) = \sum_k \theta_{jk} B_{jk}\left(t\right),\ \ j = 1, \ldots, p$ [9] are modeled with a full set of B-spline basis functions, $B_{jk}(t)$ [17]. Unlike Cox's proportional hazards model where the hazard regression coefficients, $\beta\left(t\right)$, in (1) are fixed, they are a function of time under Gray's PC-TVC model. Specifically, the coefficients are assumed to be constant only for values of $t \in \left[\tau_j, \tau_{j+1}\right),\ \ j = 0, \ldots, q$. Here $\tau_j,\ j = 1, \ldots, q$, denote the internal knots, $\tau_0 = 0$, and $\tau_{q+1} = T$ represent the maximum observed (survival or censoring) time. Under Gray's PC-TVC model, the coefficients, $\beta\left(t\right)$, are therefore right-continuous step functions of time with jumps possibly occurring at the knots $\tau_j,\ j = 1, \ldots, q$. Estimation of the regression parameters in Gray's PC-TVC model proceeds by maximizing the penalized partial likelihood, which involves a partial likelihood term as in the Cox model, plus the following penalty term: $\frac{1}{2}\lambda_j \sum_{k=2}^{q+1} \left(\theta_{jk} - \theta_{j,k-1}\right)^2$, where q is the number of internal knots for modeling the splines [9]. An essential component of the survival function estimate under Gray's PC-TVC model is based on the corresponding estimate of the cumulative baseline hazard. We extend Breslow's estimator [18] of the cumulative baseline hazard function to derive an estimator of the baseline hazard function for the TVC model. We assume that the

coefficients, β, in Breslow's formula can simply be replaced with their corresponding time-varying counterparts, $\beta(t)$. Consequently, under the TVC model (1) for the conditional hazard rate the estimated cumulative baseline hazard function is of the form:

$$\hat{\Lambda}_0(t) = \int_0^t \frac{1}{\sum_i Y_i(s)\, exp\left\{z_i'\hat{\beta}(s)\right\}} \sum_{i=1}^n dN_i(s) \quad , \tag{2}$$

where Y_i(t) is an indicator function for the i-th patient's risk status at time t (i.e., Y_i(t) = 1 if the i-th patient is in the risk set at time t, and 0 otherwise).

For Gray's PC-TVC model the formula for the estimated survival function of a patient with p-variate covariate vector, $\mathbf{z}_0$, will be:

$$\begin{aligned} \hat{S}(t|z_0) &= exp\left\{-\int_0^t d\hat{\Lambda}(s|z_0)\right\} = exp\left\{-\int_0^t exp\left\{z_0'\hat{\beta}(s)\right\} d\hat{\Lambda}_0(s)\right\} \\ &= exp\left\{-\int_0^T I(s \le t)\, exp\left\{z_0'\hat{\beta}(s)\right\} d\hat{\Lambda}_0(s)\right\}. \end{aligned} \tag{3}$$

On the log-transformed scale of the survival function we obtain:

$$log\hat{S}(t|z_0) = -\int_0^T I(s \le t) exp\left\{z_0'\hat{\beta}(s)\right\} d\hat{\Lambda}_0(s) = -\sum_{j=0}^q exp\left\{z_0'\hat{\beta}(\tau_j)\right\}\hat{\Lambda}_{0j}(t), \tag{4}$$

where

$$\hat{\Lambda}_{0j}(t) = \int_{[\tau_j,\tau_{j+1})} I(s \le t)\, d\hat{\Lambda}_0(s) = \int_{[\tau_j,\tau_{j+1})} I(s \le t) \frac{\sum_{i=1}^n dN_i(s)}{\sum_i Y_i(s)\, exp\left\{z_i'\hat{\beta}(s)\right\}} \tag{5}$$

represents a contribution to the estimated (total) cumulative baseline hazard $\hat{\Lambda}_0(t)$ corresponding to an interval $[\tau_j, \tau_{j+1})$. Since $\beta(\tau_j)$ remains constant on $[\tau_j, \tau_{j+1})$, we will make use of the following notation: $\beta_j = \beta(\tau_j)$, where β_j is a vector of length p. Given a covariate vector z_0, we thus obtain an estimate of the survival function, $S(t|z_0)$, as follows:

$$\hat{S}(t|z_0) = exp\left\{-\sum_{j=0}^q exp\left\{z_0'\hat{\beta}_j\right\}\hat{\Lambda}_{0j}(t)\right\}. \tag{6}$$

3. CONFIDENCE LIMITS BASED ON THE LOG-TRANSFORMATION

Based on (4), the formula for the variance of the log-transformed estimator of the survival function is as follows:

$$Var\left(log\hat{S}\left(t|z_0\right)\right) = Cov\left(-\sum_{j=0}^{q}\hat{\Lambda}_{0j}(t)exp\left(z_0'\hat{\beta}_j\right), -\sum_{j=0}^{q}\hat{\Lambda}_{0j}(t)exp\left(z_0'\hat{\beta}_j\right)\right) = \tag{7}$$

$$= \sum_{k=0}^{q}\sum_{l=0}^{q} Cov\left(\hat{\Lambda}_{0k}(t)exp\left(z_0'\hat{\beta}_k\right), \hat{\Lambda}_{0l}(t)exp\left(z_0'\hat{\beta}_l\right)\right) \ .$$

Note that (7) requires an estimator of the covariance which can be derived from a Taylor series approximation. We also define the following functions:

$$g(\hat{\beta}_j, t) = \hat{\Lambda}_{0j}(t)exp\left(z_0'\hat{\beta}_j\right), \quad j \in \{0, \ldots, q\}. \tag{8}$$

The vector of the corresponding partial derivatives may be evaluated as follows:

$$\frac{\partial}{\partial\hat{\beta}_j}g(\hat{\beta}_j, t) = exp\left(z_0'\hat{\beta}_j\right)\left(z_0\hat{\Lambda}_{0j}(t) + \frac{\partial}{\partial\hat{\beta}_j}\left(\hat{\Lambda}_{0j}(t)\right)\right).$$

Now, the first order Taylor series approximation of $g(\hat{\beta}_j)$ about the expected value of $\hat{\beta}_j$ (which we will denote by β_j) can be written as:

$$g(\hat{\beta}_j, t) \approx g\left(\beta_j, t\right) + \left[\frac{\partial}{\partial\hat{\beta}_j}\left(g(\hat{\beta}_j, t)\right)_{|\hat{\beta}_j=\beta_j}\right]'\left(\hat{\beta}_j - \beta_j\right) \ . \tag{9}$$

The covariance terms in (7) can be approximated at time t using the Delta method as follows:

$$Cov\left\{g(\hat{\beta}_k, t), g(\hat{\beta}_l, t)\right\} \approx W_k(t)'Cov\left(\hat{\beta}_k, \hat{\beta}_l\right)W_l(t), \tag{10}$$

where

$$W_j(t) = \left[exp(z_0'\hat{\beta}_j)\left(z_0\hat{\Lambda}_{0j}(t) + \frac{\partial}{\partial\hat{\beta}_j}\hat{\Lambda}_{0j}(t)\right)\right]_{|\hat{\beta}_j=\beta_j}, \quad j \in \{k, l\} \tag{11}$$

and

$$\frac{\partial}{\partial \hat{\beta}_j}\hat{\Lambda}_{0j}(t) = \int\limits_{[\tau_j,\tau_{j+1})} I(s \le t) \frac{-\sum_i Y_i(s)\, z_i\, exp\left\{z_i'\hat{\beta}_j\right\}}{\left\{\sum_i Y_i(s)\, exp\left\{z_i'\hat{\beta}_j\right\}\right\}^2} \sum_{i=1}^{n} dN_i(s), \quad j \in \{k,l\} \tag{12}$$

is a p-variate vector of partial derivatives of $\hat{\Lambda}_{0j}(t)$.

At time t we also have:

$$z_0\hat{\Lambda}_{0j}(t) = \int\limits_{[\tau_j,\tau_{j+1})} I(s \le t) \frac{\sum_i Y_i(s) z_0 exp\left\{z_i'\hat{\beta}_j\right\}}{\left\{\sum_i Y_i(s) exp\left\{z_i'\hat{\beta}_j\right\}\right\}^2} \sum_{i=1}^{n} dN_i(s), \tag{13}$$

so that

$$W_j(t) = \left[\int\limits_{[\tau_j,\tau_{j+1})} I(s \le t) \frac{\sum_i Y_i(s)(z_0 - z_i) exp\left\{(z_0 + z_i)'\hat{\beta}_j\right\}}{\left\{\sum_i Y_i(s)\, exp\left\{z_i'\hat{\beta}_j\right\}\right\}^2} \sum_{i=1}^{n} dN_i(s)\right]_{|\hat{\beta}_j=\beta_j}. \tag{14}$$

Consequently, the formula for the estimated variance of the predicted survival function will take the following form:

$$\hat{Var}\left(log\hat{S}(t|z_0)\right) = \sum_{k=0}^{q}\sum_{l=0}^{q} W_k(t)' Cov\left(\hat{\beta}_k, \hat{\beta}_l\right) W_l(t). \tag{15}$$

Finally, the 100(1-α)% Confidence Limits for the survival function estimated under Gray's PC-TVC model are calculated as follows:

$$exp\left(log\hat{S}(t|z_0) \pm z_{1-\alpha/2}\sqrt{\hat{Var}\left(log\hat{S}(t|z_0)\right)}\right), \tag{16}$$

where $z_{1-\alpha/2}$ denotes an upper $\alpha/2$-quantile of the standard normal distribution, $\hat{Var}\left(log\hat{S}(t|z_0)\right)$ is given by (15) and $log\hat{S}(t|z_0)$ is estimated based on equation (4).

4. CONFIDENCE LIMITS BASED ON THE LOG-(LOG)-TRANSFORMATION

On the log(-log)-scale of the estimated survival function we obtain the following:

$$log\left(-log\left(\hat{S}\left(t|z_0\right)\right)\right) = log\left(\sum_{j=0}^{q}\hat{\Lambda}_{0j}(t)exp\left(z_0'\hat{\beta}_j\right)\right). \tag{17}$$

Let us denote the complete vector of time-varying coefficient estimates from Gray's PC-TVC model by $\hat{\beta} = \left(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_q\right)$. Note that each component of the vector is itself a vector of length p (where p stands for the number of covariates being modeled by splines). Also, let $\frac{\partial}{\partial\hat{\beta}}\tilde{g}(\hat{\beta}) = \left(\frac{\partial}{\partial\hat{\beta}_0}\tilde{g}(\hat{\beta}), \frac{\partial}{\partial\hat{\beta}_1}\tilde{g}(\hat{\beta}), ..., \frac{\partial}{\partial\hat{\beta}_q}\tilde{g}(\hat{\beta})\right)$, where each of the q components of the vector of partial derivatives of $\tilde{g}(\hat{\beta})$ is itself a vector of length p. Using this notation we write at time t:

$$\tilde{g}(\hat{\beta}, t) = log\left(\sum_{j=0}^{q}\hat{\Lambda}_{0j}(t)exp\left(z_0'\hat{\beta}_j\right)\right). \tag{18}$$

Thus the k-th component of the vector of partial derivatives (being itself a vector of length p) will be:

$$\left(\frac{\partial}{\partial\hat{\beta}}\tilde{g}(\hat{\beta}, t)\right)_k = \frac{exp\left(z_0'\hat{\beta}_k\right)\left(z_0\hat{\Lambda}_{0k}(t) + \frac{\partial}{\partial\hat{\beta}_k}\left(\hat{\Lambda}_{0k}(t)\right)\right)}{\sum_{j=0}^{q}exp\left(z_0'\hat{\beta}_j\right)\hat{\Lambda}_{0j}(t)}. \tag{19}$$

It follows from (14) that:

$$\left(\frac{\partial}{\partial\hat{\beta}}\tilde{g}(\hat{\beta}, t)\right)_k = \int_{[\tau_k, \tau_{k+1})} I\left(s \le t\right)\frac{\sum_i Y_i(s)(z_0 - z_i)exp\left\{(z_0 + z_i)'\hat{\beta}_k\right\}}{\left\{\sum_{j=0}^{q}exp\left\{z_0'\hat{\beta}_j\right\}\hat{\Lambda}_{0j}(t)\right\}\left\{\sum_i Y_i(s)exp\left\{z_i'\hat{\beta}_k\right\}\right\}^2}\sum_{i=1}^{n}dN_i\left(s\right). \tag{20}$$

Let us write:

$$\tilde{W}(t) = \left[\frac{\partial}{\partial\beta}\tilde{g}(\hat{\beta}, t)\right]_{|\hat{\beta}=\beta}. \tag{21}$$

Using the first-order Taylor series approximation of the log(-log)-transformed survival function we can estimate the variance as follows:

$$\hat{Var}\left(log\left(-log\left(\hat{S}\left(t|z_0\right)\right)\right)\right) \approx \tilde{W}(t)'Var(\hat{\beta})\tilde{W}(t), \tag{22}$$

where $Var(\hat{\beta})$ is the covariance matrix of the complete vector of time-varying coefficients with the partial derivatives in expression (22) evaluated as in (20). Consequently, the 100(1-α)% confidence limits for the survival function estimated under Gray's PC-TVC model based on the log(-log) transformation of the survival function will be given by:

$$exp\left\{-exp\left\{log\left(-log\hat{S}\left(t|z_0\right)\right) \mp z_{1-\alpha/2}\sqrt{\hat{Var}\left(log\left(-log\hat{S}\left(t|z_0\right)\right)\right)}\right\}\right\}, \tag{23}$$

where $z_{1-\alpha/2}$ denotes an upper $\alpha/2$-quantile of the standard normal distribution, $log\hat{S}\left(t|z_0\right)$ is obtained from (4) and $\hat{Var}\left(log\left(-log\hat{S}\left(t|z_0\right)\right)\right)$ is estimated using (22).

5. SIMULATION STUDIES

In order to assess the accuracy of both Cox's and Gray's survival estimators we designed two simulation studies allowing for comparison of the estimated survival quantiles and probabilities of survival obtained from Cox's and Gray's model with the true underlying values. We considered scenarios that violate the assumption of proportionality. In all instances throughout this article, Gray's PC-TVC model was fitted with 10 knots selected automatically so that approximately the same number of events was observed between the successive knots,

and 4 degrees of freedom that fully specify the choice of the corresponding value of the smoothing parameter.

We have generated survival data from the piecewise-exponential distribution with two time-points allowing for a change in the hazard at .3 and .8 years. For a set of survival probabilities $\{.99., .95, .90, .75, .50, .25, .10, .05, .01\}$, the corresponding time-points were estimated using both Cox's and Gray's models based on 1000 samples of size 150. Also, for a set of time points of 3, 7, 14 and 30 days and .5, 1, 1.5 and 3 years, estimates of the corresponding probabilities of survival were calculated from each of the models. For this simulation study, all of the data are complete. The results we obtained for censored data were very similar to those for complete data. The introduction of censoring, however, leaves some quantities related to the right tail of the distribution inestimable (e.g. time points corresponding to small survival probabilities).

In the first study, one third of each sample (associated with the first covariate being an indicator function for that group) was generated with hazards of $(1.5, 1, 2)$, the second third of the sample (associated with the second covariate) was generated with the hazards reversed (i.e. $(2, 1, 1.5)$), and the baseline hazards were all taken from Exp(1). In the second study, hazards of $(2, 1, .5)$ were associated with the first covariate, those reversed $((.5, 1, 2))$ were associated with the second covariate and a constant hazard of 1 was again assumed for the baseline.

We wrote two simple Splus functions to compute the true survival quantiles and probabil-

ities for the piecewise-exponential distribution. Figure 1 (consisting of 4 panels numbered clockwise beginning from the top left quadrant) presents plots of the differences between the estimated and the true quantities (i.e. probabilities and survival quantiles respectively), as determined in both of the above studies. In both studies the survival curves were estimated at the covariate values (1,0) and (0,1) respectively, indicating a patient exhibiting the hazards specified by the first or second (i.e. reversed) set of hazards used in each example.

Panels 1 and 4 of Figure 1, based on 1000 samples, reveal that the differences between the true and estimated survival quantiles (times) were consistently smaller for Gray's model (denoted by circles in the plot). For this model the four corresponding trends in the hazard implied average departures from the true underlying quantiles of less than 20 days with the exception of the 1% quantile, for which the average departures ranged from 21 to 60 days. For the Cox model (denoted by triangles in the plot), however, departures from the true values greater than 50 days were observed for the 75, 50, 10, 5 and 1% survival quantiles. Panel 4 reveals that the estimates of the two smallest survival quantiles based on the Cox model were actually off by more than 1 year for both trends in the hazard. The magnitude of error observed was generally higher for the hazard rates of (2,1,.5) or reversed, than for those of (1.5,1,2) or reversed.

Similarly, panels 2 and 3 of Figure 1 illustrate the superior performance of Gray's PC-TVC model over that of the Cox model in terms of the accuracy of the estimated survival probabilities associated with several pre-determined time points. For 1000 samples simulated

with hazards (1.5,1,2) and (2,1,1.5) respectively, the probability estimates based on Gray's model were all within a distance of .01 from the true underlying values. For hazard rates of (2,1,.5) and (.5,1,2), estimates obtained from Gray's model exceeded the .01 distance in 3 of 18 cases with the maximum departure from the true value being .017 (associated with the time point of 6 months). Based on the Cox model, however, departures below .01 were observed in only 10 of 36 cases. In 16 of the 36 cases the magnitude of error associated with the Cox model exceeded the level of .025. The magnitude of error was again generally higher for the hazard rates (2,1,.5) or reversed, than for those of (1.5,1,2) or reversed.

The averaging effect of the Cox model is well documented in panels 2 and 3. Since the simulated hazard rates stabilized after .8 years, we observe that the departures from the true underlying values decreased dramatically after 1 year. As a result of the lack of flexibility on the part of Cox's model, however, this lead to subsequent departures in the opposite directions at the right tail of the distribution.

Results obtained from the simulation studies indicate that a high level of accuracy is maintained by the survival function estimates based on Gray's model, even in the tails of the distribution. Estimates obtained using Gray's model were generally close to the true values, while those derived from the Cox model occasionally showed large departures from the true underlying values. This resulted from a violation of the proportionality assumption in the data. The lack of precision in Cox's model was caused by the averaging of the time-varying effects, which is a built-in feature of Cox's model. In contrast, a high level of accuracy has

been maintained by Gray's survival estimator, even in the tails of the distribution.

6. UNOS DATA EXAMPLE

In this section we present a real data example comparing survival function estimators derived from Cox's and Gray's model, respectively. It features a dataset from the UNOS (United Network for Organ Sharing) database of cancer patients who underwent a liver transplant.

Here we estimate the graft survival for a subject whose covariate values are set to the median sample values. In graft survival analysis a failure is defined as an organ failure or a death of the recipient. We compare the best Cox and Gray models found for the data. The best models featured the following covariates (with corresponding sample median values listed in the parentheses): donor's anti CMV IGG result (dcmvgr, 1), indicator of whether the recipient had any prior transplant (priortx, 0), log-serum creatinine (lcreat, 0), log-total serum bilirubin (ltbili, 1.224), blood match indicator (abo.mtch, 1) and log-prothrombin time (lptp, 2.695). A summary of the modeling results may be found in Table 1. Covariates found to be significant under the best Cox model for the liver transplant graft survival of UNOS cancer patients were "lcreat", "ltbili", "dcmvgr" and "abo.mtch", with log-total serum bilirubin (ltbili) being identified as marginally non-proportional with regard to the effect on the hazard rate (p-value 0.0499). The best Gray's model included "lcreat", "lptp", "abo.mtch" and "priortx". Here the log-prothrombin time (lptp) was identified as having a highly non-proportional effect on the hazard rate (p-value 0.007).

Survival functions and 95% confidence limits estimated by the two models at the sample median covariate values are presented in Figure 2. Although the confidence bands for the two survival curves overlap (Gray's estimated survival function actually follows closely the upper confidence band estimated by the Cox's model), we can still observe a notable difference between the two survival estimates. The real data example of this section further illustrates the differences in survival estimates that might be obtained for data which does not follow the proportional hazards assumption.

7. CONCLUSIONS

Gray's piecewise-constant time-varying coefficients model for right-censored survival data is a flexible alternative to the Cox proportional hazards model in scenarios where the PH assumption may not be satisfied. The survival function estimator that we derived for this model provides a useful summary of the modeling results based on the patient's covariate values.

Simulation studies presented earlier have shown a lack of accuracy on the part of the Cox model with regard to estimating survival probabilities and predicting survival quantiles when the survival distribution does not satisfy the PH assumption.

Finally, based on Cox and Gray's model, respectively, a differing graft survival experience was demonstrated for a UNOS cancer patient after a liver transplant.

Acknowledgments

This work was supported in part by the Agency for Health Care Research and Quality (grant R01 HS09694-03) and by the Department of the Army (grant DAMD17-99-1-9536). We thank Dr. Robert J. Gray for making his own implementation of the modeling routine available to us. The survival function estimator has been written as an Splus function based on Gray's routine and is available from the authors upon request.

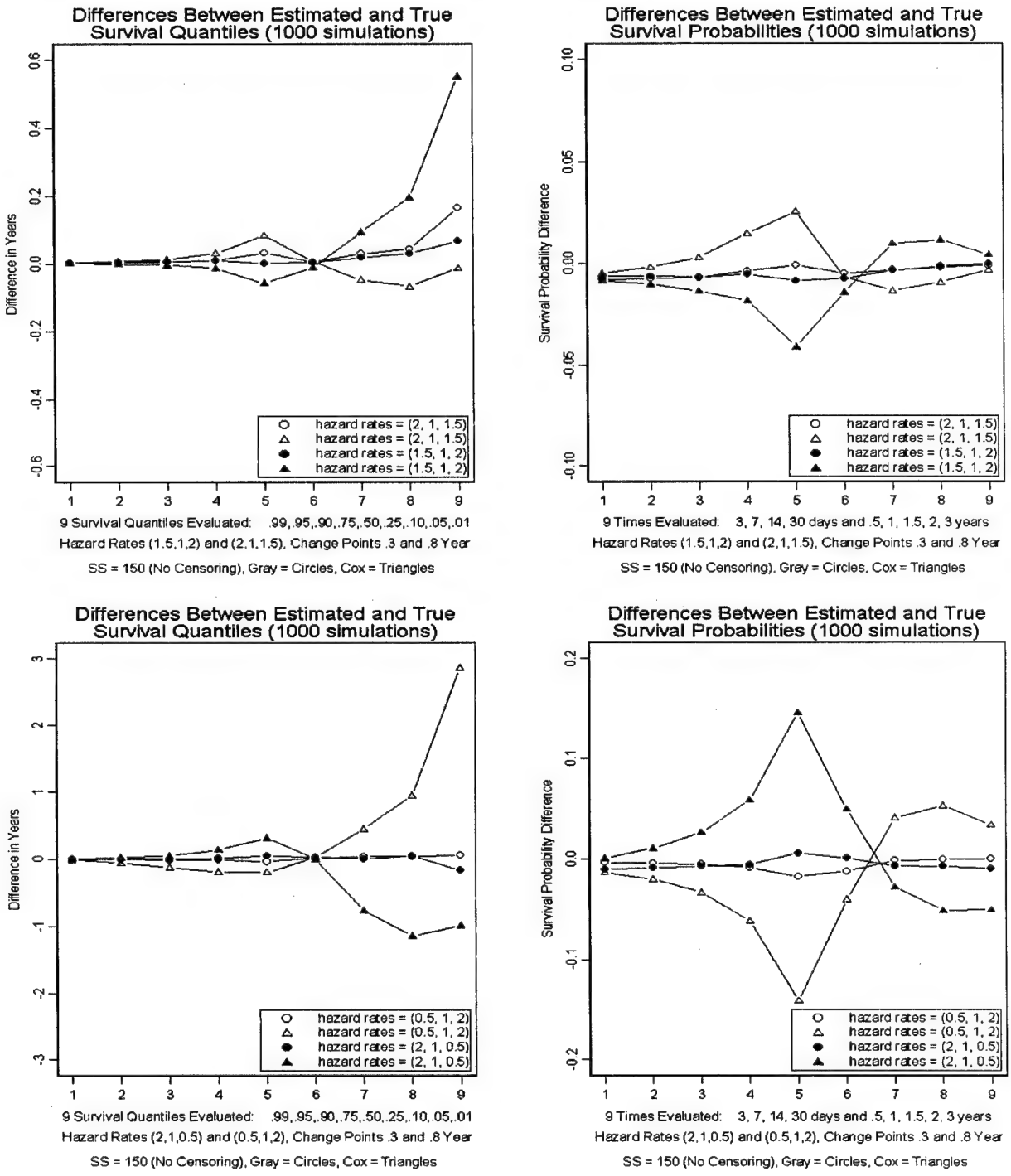

Figure 1: Simulation studies results summary

TABLE 1: Results Summary for UNOS Cancer Patients

(502 observations with 278 failures)

Covariates	Cox's Model			Gray's Model		
	Coeff	p-value	n.prop.	Coeff (Range)	p-value	n.prop.
lcreat	.266	.014	.789	(.154:.555)	.001	.277
ltbili	.182	.001	.050	-	-	-
dcmvgr	.307	.011	.936	-	-	-
abo.mtch	-1.147	.049	.642	(-2.936:.205)	.007	.142
lptp	-	-	-	(-.244:1.688)	.000	.007
priortx	-	-	-	(-5.999:3.228)	.040	.769

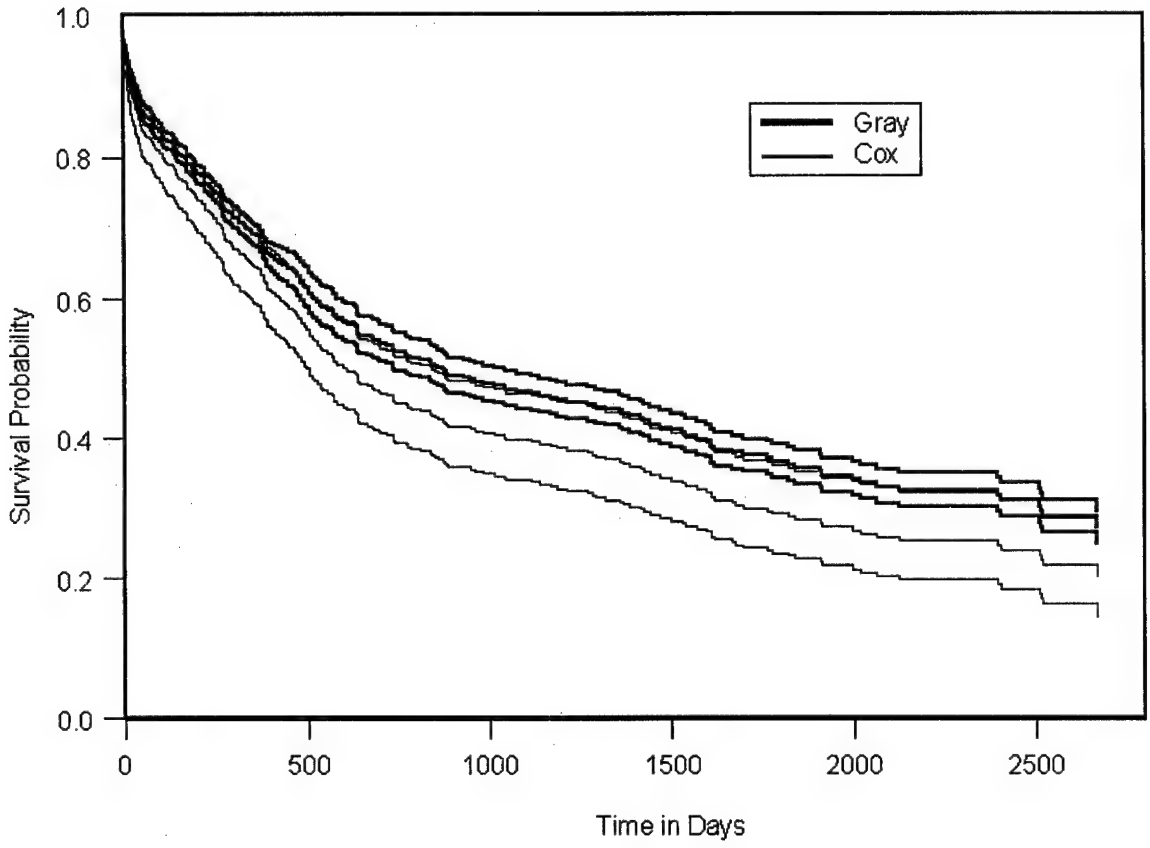

Figure 2: UNOS cancer patients post-liver transplant graft survival with 95% C.L.'s, Cox and Gray model results for a subject with median-valued covariates

References

[1] Cox DR. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society, Ser. B*, 34:187–220, 1972.

[2] Gamerman D, West M. An application of dynamic survival models in unemployment studies. *The Statistician*, 36:269–274, 1987.

[3] Zucker DM, Karr AF. Nonparametric survival analysis with time-dependent covariate effects: A penalized partial likelihood approach. *The Annals of Statistics*, 18(1):329–353, 1990.

[4] Cox DD, O'Sullivan F. Asymptotic analysis of penalized likelihood and related estimators. *The Annals of Statistics*, 18(4):1676–1695, 1990.

[5] Tsiatis AA. A large sample study of Cox's regression model. *The Annals of Statistics*, 9(1):93–108, 1981.

[6] Andersen PK, Gill RD. Cox's regression model for counting processes: A large sample study. *The Annals of Statistics*, 10:1100–1120, 1982.

[7] Gill RD. Understanding Cox's regression model: A martingale approach. *Journal of the American Statistical Asssociation*, 79:441–447, 1984.

[8] O'Sullivan F. Nonparametric estimation in the Cox model. *The Annals of Statistics*, 21(1):124–145, 1993.

[9] Gray RJ. Flexible methods for analyzing survival data using splines, with applications to breast cancer prognosis. *Journal of the American Statistical Asssociation*, 87:942–951, 1992.

[10] Gray RJ. Spline-based tests in survival analysis. *Biometrics*, 50:640–652, 1994.

[11] Hastie TJ, Tibshirani RJ. Varying coefficient models (with discussion). *Journal of the Royal Statistical Society, Ser. B*, 55(4):757–796, 1993.

[12] Sleeper LA, Harrington DP. Regression splines in the Cox model with application to covariate effects in liver disease. *Journal of the American Statistical Asssociation*, 85(412):941–949, 1990.

[13] Pettitt AN, Daud IB. Investigating time dependence in Cox's proportional hazards model. *Applied Statistics*, 39:313–329, 1990.

[14] Hess KR. Assessing time-by-covariate interactions in proportional hazards regression models using cubic spline functions. *Statistics in Medicine*, 13:1045–1062, 1994.

[15] Verweij PJM, van Houwelingen HC. Time-dependent effects of fixed covariates in Cox regression. *Biometrics*, 51(1):1550–1556, 1995.

[16] Andersen PK, Bentzon MW, Klein JP. Estimating the survival function in the proportional hazards regression model: A study of the small sample size properties. *Scandinavian Journal of Statistics*, 23:1–12, 1996.

[17] De Boor C. *A Practical Guide to Splines.* Springer Verlag, New York, 1978.

[18] Fleming TR, Harrington DP. *Counting Processes and Survival Analysis*, volume 1, section 4.3, pages 152, formula(3.29). John Wiley and Sons, Inc., One Wiley Drive, Somerset, NJ 08875, U.S.A., second edition, 1991.

[19] Breslow NE. Covariance analysis of censored survival data. *Biometrics*, 30:89–99, 1974.

[20] Hastie TJ, Tibshirani RJ. *Generalized Additive Models.* Chapman and Hall/CRC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, U.S.A., first CRC edition, 1999.

[21] O'Sullivan F. Nonparametric estimation of relative risk using splines and cross-validation. *SIAM Journal on Scientific and Statistical Computing*, 9:531–542, 1998.

APPENDIX 4. COPY OF S-PLUS CODE FOR EXECUTING GRAY'S MULTIPLE OUTCOME SURVIVAL MODEL

```
#################
# file contains s code defining functions to fit WLW (1989) type
# flexible Cox models for multiple outcome data as in Berhane and
# Weissfeld (2000?). The functions are: mcox.spline, wlwcov,
# mcox.spline.test,  spline.test cox.spline.int2
# cox.spline.plot cox.spline.mresid and cox.spline. [and to read
#jasa.data]
#
# This software comes with absolutely no guarantees.  It was modified
# from the software by Robert Gray (1992), Harvard Univeristy.
# You have permission to
# use it for any noncommercial purpose, and to modify it as needed.
#
#
#################
#
# An S-plus function to fit a cox.spline model on each of the multiple
# margins and calculate the variance covariance matrix (as in Berhane
# and Weissfeld, 2000?) for simultaneous Wald type inference across
# the margins.
#
# Input variables are:
#
#  gind          --- Outcome type indicator vector
#  model.type    --- as in cox.spline (only "a" [for additive] allowed)
#  time          --- vector of failure times (sorted by outcome type)
#  status              --- 1 if failure, 0 if censored (by outcome type)
#  spline.cov    --- list of outcome-specific matrices of covaraites
#                to be modeled with splines
#  linear.cov    --- list of outcome-specific matrices of covaraites
#                to be modeled as linear terms
#  strata        --- stratification variable (numeric,
#                character, or category) sorted by outcome type
#                Default is all observations in one strata
#  df            --- matrix of degrees of freedom for the spline fits
#                                one column per outcome type.
#  Note: Ignored if smooth.param is specified.
#  Default is 3 for nonlinearity (about 4 total) for each variable when
#  model.type="a", 3 for each variable when model.type="t", and 8 for the
#  tensor product term when model.type="i".
#  If length of df is not of the appropriate length (1 for "i", the number
#  of columns in spline.cov for "a" or "t"), a warning message is printed,
#  and df replicated to the appropriate length.
#
#  nknot         --- number of knots for spline variables (length 1 for
#                    "a" or "t", since the same value is used for all the
#                     covariates and outcome types, and length 2 for "i").
#  spline.knot   --- locations of knots for splines
#  Note:         Each set of knots must be an augmented
#  knot sequence of length nknot+6, with something <= min for the first
#  three entries, then the
#  interior knots, then something >= the max for the next 3 entries. If
#  "a" then a set of knots is needed for each column of spline.cov. If
#  "i", 2 sets of knots are needed, and if "t" only one is needed. Default
#  is for the program to choose knot locations based on equal numbers of
#  observations between knots. (Default for now)
#
```

```
#  smooth.opt    --- smoothing parameter option.
#  Note:         If smooth.param is not specified, the smoothing parameters
#  in the penalty function are chosen to give the specified degrees
#  of freedom.  If smooth.opt>0 smoothing parameters adjusted after every
#  iteration, if <= 0 they are calculated after the first iteration only,
#  and then held fixed.  This can speed convergence, but the degrees of
#  freedom at the final estimates may not match the specified df.
#
#       smooth.param    --- matrix of smoothing parameters.
#         Note: If specified must have dimension equal to
#       G (# outcome type) columns and raws equal to the number
#       of columns in spline.cov ("a", "t") or 1 ("i").
#       If specified, the smoothing parameters are kept fixed at these
#       values throughout the iteration.
#
# Output variables are:
#
#       m.out    --- A list of results from marginal fits of
#                of every outcome type.
#
#       mbeta    --- vector of regression parameter estimates
#                sorted by outcome types.
#
#       Qhat     --- Variance-covariance matrix between outcome types
#                for simultaneous inference.
#
#     test.list --- summary of simultaneous inference across outcome
#           types for overall significance and for n0n-linearity
#
mcoxmp.spline_function(gind, m.type, time, status, spline.cov, linear.cov,
      tstrata,dof, nknot, spline.knot, smooth.opt = 1, smooth.param,
      output.opt = "tests", nest, maxiter = 30, eps = 0.0001,
      rescale = T, ord = 0)
{
      g <- length(unique(gind))
      m.out <- vector("list", g)
      mbeta <- NULL
      w.time <- cbind(gind, time)
      w.status <- cbind(gind, status)
      if(!missing(spline.cov))
            wspline.cov <- cbind(gind, spline.cov)
      if(!missing(linear.cov))
            wlinear.cov <- cbind(gind, linear.cov)
      if(!missing(tstrata))
            w.strata <- cbind(gind, tstrata)
      for(i in 1:g) {
      t.time <- w.time[gind == i, -1]
                o.time <- order(t.time)
                i.time <- t.time[o.time]
                id <- rep(1:length(i.time),1)
                o.id <- id[o.time]
                t.status <- w.status[gind == i, -1]
                i.status <- t.status[o.time]
                if(!missing(spline.cov)) {
                        if(!missing(linear.cov)) {
                                tspline.cov <- as.matrix(
                                        wspline.cov[gind == i, -1])
```

```
                ispline.cov <- tspline.cov[o.time,]
                tlinear.cov <- as.matrix(
                        wlinear.cov[gind == i, -1])
                ilinear.cov <- tlinear.cov[o.time,]
        }
        else {
                tspline.cov <- as.matrix(
                        wspline.cov[gind == i, -1])
                ispline.cov <- tspline.cov[o.time,]
                #print(nrow(ispline.cov))
        }
}
else {
        tlinear.cov <- as.matrix(wlinear.cov[gind == i, -1])
        ilinear.cov <- tspline.cov[o.time,]
        #print(nrow(ilinear.cov))
}
t.strata <- w.strata[gind == i, -1]
i.strata <- t.strata[o.time]
#cat("ready for model\n")
if(!missing(smooth.param)) {
        if(!missing(spline.cov)) {
                if(missing(linear.cov)) {
                        m.out[[i]] <- cox.spline(model.type
                                =m.type,time=i.time,
                                status=i.status,
                                spline.cov = ispline.cov,
                                strata = i.strata,
                                df = dof, nknot = nknot,
                                smooth.opt=1,smooth.param=
                                smooth.param[, g], output.opt
                                 = "tests", maxiter = 30, eps
                                 = 0.0001, rescale = T, ord = 0
                                )
                }
                else {
                        m.out[[i]] <-cox.spline(model.type=
                                m.type,time = i.time,
                                status=i.status,
                                linear.cov = ilinear.cov,
                                spline.cov = ispline.cov,
                               strata = i.strata,
                                df=dof,nknot=nknot,
                                smooth.opt=1,smooth.param=
                                smooth.param[, g],
                                output.opt = "tests",
                                maxiter = 30, eps=0.0001,
                                rescale = T, ord = 0)
                }
        }
        else {
                m.out[[i]] <-cox.spline(model.type=m.type,
                        time=i.time, i.status,
                        linear.cov = ilinear.cov,
                        strata = i.strata,df = dof,
                        nknot = nknot, smooth.opt = 1,
                        smooth.param = smooth.param[, g],
```

```
                                        output.opt = "tests",
                                        maxiter = 30, eps = 0.0001, rescale = T,
                                        ord = 0)
                                }
                        }
                        else {
                                if(!missing(spline.cov)) {
                                        if(!missing(linear.cov)) {
                                                #cat("spl,lin\n")
                                                m.out[[i]] <- cox.spline(model.type
                                                        = m.type, time=i.time,
                                                        status=i.status,
                                                        linear.cov = ilinear.cov,
                                                        spline.cov = ispline.cov,
                                                        strata = i.strata,
                                                        df = dof,nknot = nknot,
                                                        smooth.opt = 1,
                                                        output.opt = "tests", maxiter
                                                         = 30, eps = 0.0001, rescale =
                                                        T, ord = 0)
                                        }
                                        else {
                                                #cat("spl.lin.mis\n")
                                                m.out[[i]] <- cox.spline(
                                                        model.type=m.type,
                                                        time=i.time,
                                                        status=i.status,
                                                        spline.cov = ispline.cov,
                                                        strata = i.strata,
                                                        df = dof,
                                                        nknot = nknot,
                                                        smooth.opt = 1,
                                                        output.opt = "tests",
                                                        maxiter = 30, eps = 0.0001,
                                                        rescale = T, ord = 0)
                                        }
                                }
                                else {
                                        #cat("spline missing\n")
                                        m.out[[i]] <-cox.spline(model.type=m.type,
                                                time=i.time,
                                                strata = i.status,
                                                linear.cov = ilinear.cov,
                                                strata = i.strata,df=dof,
                                                nknot=nknot, smooth.opt=1,
                                                output.opt = "tests", maxiter = 30,
                                                eps = 0.0001, rescale = T, ord = 0)
                                }
                        }
                        mbeta <- c(mbeta, m.out[[i]]$coef)
                        tw _ m.out[[i]]$w
                        m.out[[i]]$w _ tw[order(o.id),]
                twp _ m.out[[i]]$wp
                m.out[[i]]$wp _ twp[order(o.id),]
        }
        gp <- length(mbeta)
        nump <- gp/g
```

```
        p <- nump
        Qhat <- matrix(rep(0, gp * gp), ncol = gp)
        for(i in 1:g) {
              for(j in 1:g) {
                    i.ind <- (((i - 1) * p) + 1):(((i - 1) * p) + p)
                    j.ind <- (((j - 1) * p) + 1):(((j - 1) * p) + p)
                    Qhat[i.ind, j.ind] <- wlwcov.wt(m.out[[i]]$d2i,
                                m.out[[j]]$d2i,
                                m.out[[i]]$var,
                          m.out[[j]]$var,
                                m.out[[i]]$wp, m.out[[j]]$wp,
                                m.out[[i]]$coef, m.out[[j]]$coef)$dij
              }
        }
          #cat('Qhat is:\n ')
          mcox.out_mcox.spline.intm(m.out,Qhat)
          return(Qhat,m.out, mbeta, g, mcox.out)
}

wlwcov.wt_function(pvi,pvj,uvi,uvj,wi,wj,betai,betaj){
# An S-plus auxilliary function to calculate the middle (meat) part of the
# variance-covariance for the multiple outcome data (called by
# mcox.spline)
#
# Input variables are:
#
#       vi, vj          --- Var-Cov matrices for ith and jth outcome types
#       wi, wj          --- W matrices for ith and jth outcome types as in
#                           Berhane and Weissfeld (2000?, Eqn. 10)
#
# Output variables are:
#
#       Phi             --- Middle part of the var-cov matrix between ith
#                           and jth outcome types
#       dij             --- Var-cov matrix for ith and jth outcome types
#
nobs _ nrow(na.omit(wi))
phi_t(wi)%*%wj
#phi_phi/nobs
uvi_solve(uvi)
uvj_solve(uvj)
uvi_(uvi+t(uvi))/2
uvj_(uvj+t(uvj))/2
vi_pvi%*%uvi%*%pvi
vj_pvj%*%uvj%*%pvj
dij_vi%*%phi%*%vj
#dij_dij/nobs
return(phi,dij)
}


mcox.spline.intm _ function(zlist,Qhat) {
# zlist is a list pf the lists created by cox.spline on several outcomes
# (margins) but with the same variables and knots (options "a" and "t" only).
if (any(zlist[[1]]$nvar != zlist[[2]]$nvar) |
    (length(zlist[[1]]$coef) != length(zlist[[2]]$coef)))
     stop("models must include the same variables")
```

```
if (zlist[[1]]$type == "a" & zlist[[2]]$type == "a") {
  nsplin _ zlist[[1]]$nvar[2]
  nknot _ length(zlist[[1]]$knot)/nsplin-6
  nfixed _ zlist[[1]]$nvar[1]
 if (nfixed>0) {
    g_length(zlist)
    #cat("g is: ")
    #print(g)
    out.fn _ vector("list",nfixed)
    out.fr _ vector("list",nfixed)
    #names(out.fn) _ names(z1$coef)[1:nfixed]
    names(out.fr) _ names(out.fn)
    for (i in 1:nfixed) {
      np1.t _ length(zlist[[1]]$coef)
      np.t_np1.t*g
      theta.f_NULL
      np.f_0
      for(j in 1:g){
      theta.fup_ zlist[[j]]$coef[i]
      theta.f_c(theta.f,theta.fup)
      np.f_np.f+1
      }
      v.f _ matrix(0,nrow=np.f,ncol=np.f)
      h.f _ v.f
      q.f_v.f
      npt_0
      for(j in 1:g){
      v.f[j,j]_zlist[[j]]$var[i,i]
      h.f[j,j]_zlist[[j]]$d2i[i,i]
      q.f[j,j]_Qhat[npt+i,npt+i]
      npt_npt+np1.t
      }
      ct.f _ diag(np.f)
      out.fn[[i]]_spline.test.naive(theta.f,h.f,v.f,q.f,ct.f)
      out.fr[[i]]_spline.test.rob(theta.f,h.f,v.f,q.f,ct.f)
      }
  }
  else {
  out.fn _ NULL
  out.fr_NULL
  }

  if (nsplin>0) {
    g_length(zlist)
    #cat("g is: ")
    #print(g)
    out.an _ vector("list",nsplin)
    out.ln_vector("list",nsplin)
    out.ar _ vector("list",nsplin)
    out.lr_vector("list",nsplin)
    names(out.an) _ names(zlist[[1]]$test)
    names(out.ln) _ names(zlist[[1]]$test)
    names(out.ar) _ names(zlist[[1]]$test)
    names(out.lr) _ names(zlist[[1]]$test)
    for (i in 1:nsplin) {
      l1 _ nfixed+nsplin+(i-1)*(nknot+2)+1
      l2 _ l1+nknot+1
```

```
np1.t _ length(zlist[[1]]$coef)
np.t_np1.t*g
theta.a_NULL
np.a_0
for(j in 1:g){
theta.up_ zlist[[j]]$coef[c(nfixed+i,l1:l2)]
theta.a_c(theta.a,theta.up)
np.a_np.a+length(theta.up)
}
np1.a_np.a/g
v.a _ matrix(0,nrow=np.a,ncol=np.a)
h.a _ v.a
q.a_v.a
lim.at_NULL
npa.j_0
npt.j_0
for(j in 1:g){
lim.tt_c(npt.j+nfixed+i,(npt.j+l1):(npt.j+l2))
lim.t_c(nfixed+i,l1:l2)
lim.a_c((npa.j+1):(npa.j+np1.a))
lim.at_c(lim.at,lim.a)
v.a[lim.a,lim.a]_zlist[[j]]$var[lim.t,lim.t]
h.a[lim.a,lim.a]_zlist[[j]]$d2i[lim.t,lim.t]
npa.k_0
npt.k_0
for(k in 1:g){
          mult.ka_((k-1)*npa.k)
          mult.ktt_((k-1)*npt.k)
          mult.ja_((j-1)*npa.j)
          mult.jtt_((j-1)*npt.j)
          klim.a_c((mult.ka+1):(mult.ka+np1.a))
          klim.tt_c(mult.ktt+nfixed+i, (mult.ktt+l1):(mult.ktt+l2))
          jlim.a_c((mult.ja+1):(mult.ja+np1.a))
          jlim.tt_c(mult.jtt+nfixed+i, (mult.jtt+l1):(mult.jtt+l2))
          q.a[jlim.a,klim.a]_Qhat[jlim.tt,klim.tt]
          npa.k_npa.k+np1.a
          npt.k_npt.k+np1.t
          }
npa.j_npa.j+np1.a
npt.j_npt.j+np1.t
}
ct.a _ diag(np.a)
theta.l_NULL
np.l_0
for(j in 1:g){
theta.up_zlist[[j]]$coef[c(l1:l2)]
theta.l_c(theta.l,theta.up)
np.l_np.l+length(theta.up)
}
np1.l_np.l/g
v.l _ matrix(0,nrow=np.l,ncol=np.l)
h.l _ matrix(0,nrow=np.l,ncol=np.l)
q.l_matrix(0,nrow=np.l,ncol=np.l)
lim.lt_NULL
npl.j_0
npt.j_0
for(j in 1:g){
```

```
        lim.tt_c((npt.j+l1):(npt.j+l2))
        lim.t_c(l1:l2)
        lim.l_c((npl.j+1):(npl.j+npl.l))
        lim.lt_c(lim.lt,lim.l)
        v.l[lim.l,lim.l]_zlist[[j]]$var[l1:l2,l1:l2]
        h.l[lim.l,lim.l]_zlist[[j]]$d2i[l1:l2,l1:l2]
        npl.k_0
        npt.k_0
        for(k in 1:g){
                  mult.kl_((k-1)*npl.k)
                  mult.ktt_((k-1)*npt.k)
                  mult.jl_((j-1)*npl.j)
                  mult.jtt_((j-1)*npt.j)
                  klim.l_c((mult.kl+1):(mult.kl+npl.l))
                  klim.tt_c((mult.ktt+l1):(mult.ktt+l2))
                  jlim.l_c((mult.jl+1):(mult.jl+npl.l))
                  jlim.tt_c((mult.jtt+l1):(mult.jtt+l2))
                  q.l[jlim.l,klim.l]_Qhat[jlim.tt,klim.tt]
                  npl.k_npl.k+npl.l
                  npt.k_npt.k+npl.t
          }
        npl.j_npl.j+npl.l
        npt.j_npt.j+npl.t
        }
        ct.l _ diag(np.l)
        out.an[[i]]_spline.test.naive(theta.a,h.a,v.a,q.a,ct.a)
        out.ln[[i]]_spline.test.naive(theta.l,h.l,v.l,q.l,ct.l)
        out.ar[[i]]_spline.test.rob(theta.a,h.a,v.a,q.a,ct.a)
        out.lr[[i]]_spline.test.rob(theta.l,h.l,v.l,q.l,ct.l)
    }
  }
  else {
  out.an _ NULL
  out.ln _ NULL
  out.ar _ NULL
  out.lr _ NULL
  }
}
else stop("lists must be of same type")
return(out.an,out.ln,out.ar,out.lr,out.fn,out.fr)
}

spline.test.naive_function(theta, h, v, q, ct)
{
        # routine for wald like test for the gen lin hypoth ctxtheta=0
        #using estimates from a penalized likelihood
        # theta_parameter estimates
        # h=inverse of second derivative matrix from penalized likelihood
        # v=var-cov matrix of parameter estimates
        # ct=contrast matrix.  tests ct theta = 0
        # first compute cthct' ctvct'
        h <- ct %*% h %*% t(ct)
        v <- ct %*% v %*% t(ct)
        q <- ct %*% q %*% t(ct)
        theta.t <- ct %*% theta
        # calc eigenvalues
        # first correct h for possible nonsymmetry due to roundoff:
```

```
        h <- (t(h) + h)/2
        h <- solve(chol(h))
        q <- (t(q) + q)/2
        #q <- solve(chol(q))
        #q <- solve(q)
        theta <- t(h) %*% theta
        stat <- t(theta) %*%  theta
        #theta.t <- t(q) %*% theta.t
        #stat <- t(theta.t) %*% theta.t
        #stat <- t(theta.t) %*% q %*% theta.t
        h <- t(h) %*% v %*% h
        h <- eigen(h)$value
        k <- length(theta)
        df <- rep(1, k)
        v <- rep(0, k)
        v <- .Fortran("dwch",
                as.double(h),
                as.double(v),
                as.integer(df),
                as.integer(k),
                as.double(stat),
                double(1),
                as.integer(df))[[6]]
        df <- sum(h)
        v <- c(stat, v, df)
        names(v) <- c("stat", "pv", "df")
        v
}

spline.test.rob_function(theta, h, v, q, ct)
{
        # routine for wald like test for the gen lin hypoth ctxtheta=0
        #using estimates from a penalized likelihood
        # theta_parameter estimates
        # h=inverse of second derivative matrix from penalized likelihood
        # v=var-cov matrix of parameter estimates
        # ct=contrast matrix.  tests ct theta = 0
        # first compute cthct' ctvct'
        h <- ct %*% h %*% t(ct)
        v <- ct %*% v %*% t(ct)
        q <- ct %*% q %*% t(ct)
        theta.t <- ct %*% theta
        # calc eigenvalues
        # first correct h for possible nonsymmetry due to roundoff:
        h <- (t(h) + h)/2
        h <- solve(chol(h))
        q <- (t(q) + q)/2
        q <- solve(chol(q))
        #q <- chol(q)
        #q <- solve(q)
        #theta <- t(h) %*% theta
        #stat <- t(theta) %*%  theta
        theta.t <- t(q) %*% theta.t
        stat <- t(theta.t) %*% theta.t
        #stat <- t(theta.t) %*% q %*% theta.t
        h <- t(h) %*% v %*% h
        h <- eigen(h)$value
```

```
        #h <- t(h) %*% q %*% h
        #h <- eigen(h)$value
        k <- length(theta)
        df <- rep(1, k)
        v <- rep(0, k)
        v <- .Fortran("dwch",
               as.double(h),
               as.double(v),
               as.integer(df),
               as.integer(k),
               as.double(stat),
               double(1),
               as.integer(df))[[6]]
        df <- sum(h)
        #df <- k
        #v <- 1-pchisq(stat,df)
        v <- c(stat, v, df)
        names(v) <- c("stat", "pv", "df")
        v
}
```

```
mgrayp1.simu_function(n,m,pcens,bhaz,alpha=0.25,dfr,nknots,conf.level){
cens_runif(2*n,0,1)
cens_ifelse(cens>pcens,1,0)
#z1 <- 100*runif(n)
#z2 <- 100*runif(n)
z1_runif(n)
z1_(z1-mean(z1))/(sqrt(var(z1)))
#z1_seq(from=-1.71,to=1.71,length=n)
z2_runif(n)
z2_(z2-mean(z2))/(sqrt(var(z2)))
#z1 <- ifelse(z1<0.5,1,0)
#z2 <- ifelse(z2<0.5,1,0)
m.mat <- matrix(0,1,2)
#lamda1 <- bhaz*exp( 3 * z1 + 3 * z2)
#lamda2 <- bhaz*exp( 3 * z1 + 5 * z2)
#lamda1 <- bhaz*exp( 3 * z1 )
#lamda2 <- bhaz*exp( 3 * z1 )
lamda1 <- bhaz
lamda2 <- bhaz
count_0
mcox.fit.list_vector("list",m)
for(i in 1:m){

bivdata <- simu.bivexp(n,lamda1,lamda2,alpha)
biv.data <- as.data.frame(cbind(c(bivdata$x,bivdata$y),c(z1,z1),
            c(z2,z2),cens,c(rep(1,n),rep(2,n)),rep(1,2*n)))
names(biv.data) <- c("eventt","z1","z2","cens","groups","stratas")
#cat("dim(biv.data) is:" )
#print(dim(biv.data))
#print(biv.data[1:10,])
```

```
#names(biv.data) <- c("x","y","z1","z2","cen1","cen2")
cv1_biv.data$z1
cv2_biv.data$z2
#fail <- as.data.frame(cbind(cens))
#linear.cov <- array(c(cv1,cv2),dim=c(400,2))
#spline.cov <- as.matrix(bivexp.dt1[,c("cv1","cv2")])
##cov <- as.data.frame(cbind(cv1,cv2))
##names(cov)_c("x1","x2")
cov <- as.data.frame(cbind(cv1,cv2))
names(cov)_c("x1","x2")
#linear.cov <- array(c(cv1))
#spline.cov <- array(c(cv1,cv2),dim=c(400,2))
cov_as.matrix(cov)
mcox.fit_mcoxmp.spline(gind=biv.data$groups,m.type="a",
        time=biv.data$eventt,status=biv.data$cens,
        spline.cov=cov[,1],tstrata=biv.data$stratas,
        dof=dfr[1],nknot=nknots)
out.an_as.vector(mcox.fit$mcox.out$out.an$spl1[2])
out.ar_as.vector(mcox.fit$mcox.out$out.ar$spl1[2])
##cat("spl1[2] is:")
##print(c(out.an,out.ar))
##print(ifelse(out.an < conf.level,1,0))
##print(as.numeric(ifelse(out.ar < conf.level,1,0)))
m.mat[1,1]_m.mat[1] + as.vector(ifelse(out.an < conf.level,1,0))
m.mat[1,2]_m.mat[2] + as.vector(ifelse(out.ar < conf.level,1,0))
#m.mat[1,3]_m.mat[1,3] + ifelse(mcox.fit$mcox.out$out.ln$x1[2] < 0.05,1,0)
#m.mat[1,4]_m.mat[1,4] + ifelse(mcox.fit$mcox.out$out.lr$x1[2] < 0.05,1,0)
mcox.fit.list[[i]]_mcox.fit$mcox.out
count_count+1
#x <- as.matrix(bivexp.dt1[,c("cv1","cv2")])
#z <- cox.spline("a",tevt,cens1,spline.cov=x,strata=strata1,
#      df=c(3,3),nknot=10)
}
##cat("m.mat is:")
##print(m.mat)
##print(is.matrix(m.mat))
m.mat_as.data.frame(m.mat)
names(m.mat)_c("g1a.n","g1a.r")
m.mat_m.mat/count
return(count,m.mat,mcox.fit.list,mcox.fit$mcox.out)
}

simu.bivexp <- function(n,lamda1,lamda2,alpha = 0.25)
{
# n ..... number of observations per strata
# x,y.. ..... event time for the bivariate
# lamda1, lamda2 corresponding hazards

#u1 <- rep(0,n)
#u2 <- rep(0,n)
#x <- rep(0,n)
u1 <- runif(n)
u2 <- runif(n)
x <- -log(1 - u1)/lamda1
a <- rep(0,n)
b <- rep(0,n)
c <- rep(0,n)
```

```
v <- alpha*(2*exp(-x*lamda2)-1)
a <- v
b <- -(1+v)
c <- (1-u2)
w2 <- (-b - sqrt(b**2-4*a*c))/(2*a)
y <- -log(1-w2)/lamda2
return(x,y)
}
```

```
#################
# file contains s code defining functions spline.test cox.spline.int2
# cox.spline.plot cox.spline.mresid cox.spline, and to read jasa.data
#
# This software comes with absolutely no guarantees.  You have permission to
# use it for any noncommercial purpose, and to modify it as needed.
# Copyright 1992 by Robert Gray
#
#################

spline.test _ function(theta,h,v,ct) {
# routine for wald like test for the gen lin hypoth ctxtheta=0
#using estimates from a penalized likelihood
# theta_parameter estimates
# h=inverse of second derivative matrix from penalized likelihood
# v=var-cov matrix of parameter estimates
# ct=contrast matrix.  tests ct theta = 0
# first compute cthct' ctvct'
h _ ct%*%h%*%t(ct)
v _ ct%*%v%*%t(ct)
theta _ ct%*%theta
# calc eigenvalues
# first correct h for possible nonsymmetry due to roundoff:
h _ (t(h)+h)/2
h _ solve(chol(h))
theta _ t(h)%*%theta
stat _ t(theta)%*%theta
h _ t(h)%*%v%*%h
h _ eigen(h)$value
k _ length(theta)
df _ rep(1,k)
v _ rep(0,k)
v _
.Fortran("dwch",as.double(h),as.double(v),as.integer(df),as.integer(k),
        as.double(stat),double(1),as.integer(df))[[6]]
df _ sum(h)
v _ c(stat,v,df)
names(v) _ c("stat","pv","df")
v
}

cox.spline.int2 _ function(z1,z2) {
# z1 and z2 are lists created by cox.spline on different subgroups
```

```
# but with the same variables and knots (options "a" and "t" only).
if (any(z1$nvar != z2$nvar) | (length(z1$coef) != length(z2$coef)))
     stop("models must include the same variables")
if (z1$type == "a" & z2$type == "a") {
  nsplin _ z1$nvar[2]
  nknot _ length(z1$knot)/nsplin-6
  nfixed _ z1$nvar[1]
  if (nfixed>0) {
    outf _ vector("list",nfixed)
    names(outf) _ names(z1$coef)[1:nfixed]
    for (i in 1:nfixed) {
      stat _ (z1$coef[i]-z2$coef[i])/sqrt(z1$var[i,i]+z2$var[i,i])
      pv _ 1-pchisq(stat*stat,1)
      outf[[i]] _ c(stat,pv)
      names(outf[[i]]) _ c("stat","pv")
    }
  }
  else outf _ NULL
  if (nsplin>0) {
    out _ vector("list",nsplin)
    names(out) _ names(z1$test)
    for (i in 1:nsplin) {
      l1 _ nfixed+nsplin+(i-1)*(nknot+2)+1
      l2 _ l1+nknot+1
      theta _ c(z1$coef[c(nfixed+i,l1:l2)],z2$coef[c(nfixed+i,l1:l2)])
      np _ length(theta)
      np2 _ np/2
      h _ matrix(0,nrow=np,ncol=np)
      h[1:np2,1:np2] _ z1$d2i[c(nfixed+i,l1:l2),c(nfixed+i,l1:l2)]
      h[(np2+1):np,(np2+1):np] _ z2$d2i[c(nfixed+i,l1:l2),c(nfixed+i,l1:l2)]
      v _ matrix(0,nrow=np,ncol=np)
      v[1:np2,1:np2] _ z1$var[c(nfixed+i,l1:l2),c(nfixed+i,l1:l2)]
      v[(np2+1):np,(np2+1):np] _ z2$var[c(nfixed+i,l1:l2),c(nfixed+i,l1:l2)]
      ct _ cbind(diag(np2),-diag(np2))
      out[[i]] _ spline.test(theta,h,v,ct)
    }
  }
  else out _ NULL
  c(outf,out)
}
else if (z1$type == "t" & z2$type == "t") {
  nsplin _ z1$nvar[2]
  nfixed _ z1$nvar[1]
  nb _ (length(z1$coef)-nfixed)/nsplin
  if (nfixed>0) {
    outf _ vector("list",nfixed)
    names(outf) _ names(z1$coef)[1:nfixed]
    for (i in 1:nfixed) {
      stat _ (z1$coef[i]-z2$coef[i])/sqrt(z1$var[i,i]+z2$var[i,i])
      pv _ 1-pchisq(stat*stat,1)
      outf[[i]] _ c(stat,pv)
      names(outf[[i]]) _ c("stat","pv")
    }
  }
  else outf _ NULL
  if (nsplin>0) {
    out _ vector("list",nsplin)
```

```
    names(out) _ names(z1$test)
    for (i in 1:nsplin) {
      l1 _ nfixed+(i-1)*nb+1
      l2 _ l1+nb-1
      theta _ c(z1$coef[l1:l2],z2$coef[l1:l2])
      np _ length(theta)
      np2 _ np/2
      h _ matrix(0,nrow=np,ncol=np)
      h[1:np2,1:np2] _ z1$d2i[l1:l2,l1:l2]
      h[(np2+1):np,(np2+1):np] _ z2$d2i[l1:l2,l1:l2]
      v _ matrix(0,nrow=np,ncol=np)
      v[1:np2,1:np2] _ z1$var[l1:l2,l1:l2]
      v[(np2+1):np,(np2+1):np] _ z2$var[l1:l2,l1:l2]
      ct _ cbind(diag(np2),-diag(np2))
      out[[i]] _ spline.test(theta,h,v,ct)
    }
  }
  else out _ NULL
  c(outf,out)
}
else stop("lists must be of same type")
}

cox.spline.mresid _ function(time,status,spline.cov,coef,spline.knot,
linear.cov,strata)  {

no _ length(time)
if (length(status) != no) stop("status wrong length")

# note that splin covs are not first in cov matrix if linear covs are:
if (missing(linear.cov)) {
  linear.cov _ as.matrix(spline.cov)
  if (no != nrow(linear.cov)) stop("nrow mismatch in spline.cov")
  nsx _ ncol(linear.cov)
}
else {
  linear.cov _ as.matrix(linear.cov)
  if (no != nrow(linear.cov)) stop("nrow mismatch in linear.cov")
  spline.cov _ as.matrix(spline.cov)
  if (no != nrow(spline.cov)) stop("nrow mismatch in spline.cov")
  nsx _ ncol(spline.cov)
  linear.cov _ cbind(linear.cov,spline.cov)
}
nfx _ ncol(linear.cov)

if (missing(strata)) {
  strata _ rep(1,no)
} else {
  if (length(strata) != no) stop("strata vector is the wrong length")
}
strata _ codes(factor(strata))

nomiss _  !(is.na(time) | is.na(status) | is.na(strata) |
            is.na(linear.cov %*% rep(1,nfx)))
# check status variable
  if (any(status[nomiss] != 0 & status[nomiss] != 1))
          stop("invalid status values")
```

```
time _ time[nomiss]
status _ status[nomiss]
strata _ strata[nomiss]
linear.cov _ linear.cov[nomiss,]

no _ length(time)
nostr _ table(strata)
nstr _ length(nostr)
nostr _ cumsum(nostr)

nkl _ length(spline.knot)/nsx - 6
nknot <- (length(coef)-nfx)/nsx - 2
nknot _ length(spline.knot)/nsx - 6

elp _ .Fortran("cests",as.double(linear.cov),
                       as.integer(no),as.integer(nfx),as.integer(nsx),
                       as.double(coef),as.integer(length(coef)),
                       as.integer(nknot),
                       as.double(spline.knot),as.integer(nkl),double(no))[[10]]

o _ order(strata,time)
mresid _ .Fortran("mresd",as.double(time[o]),as.double(status[o]),
                          as.double(elp[o]),as.integer(no),
                          as.integer(nstr),as.integer(nostr))[[3]]
mresid[o] _ mresid
mresid
}

cox.spline.plot _ function(z,ncovplot,xlab,main="",
   ylab,zlab="Log Hazard Ratio",ylim,plotopt=T,pvar=T,knots=F,font=3,lwd=1,...)
{
if (z$type == "i") {           # plot for interaction
  x_sort(unique(z$est[,1]))
  y_sort(unique(z$est[,2]))
  u_z$est[,3]
  dim(u)_c(length(y),length(x))
  u_t(u)
  if (plotopt) {
    if (missing(xlab)) xlab _ dimnames(z$est)[[2]][1]
    if (missing(ylab)) ylab _ dimnames(z$est)[[2]][2]
    if (!exists(".Device")) stop("Graphics device must be active")
    persp(x,y,u,xlab=xlab,ylab=ylab,zlab=zlab,font=font,lwd=lwd,...)
    mtext(main,cex=1.5)
  }
}
else if (z$type == "t") {     # plots for tvc's
  ec _ 2*ncovplot
  if (pvar) {
    u _ ifelse(z$est[,ec+1]<0,0,z$est[,ec+1])
    u _ 2*sqrt(u)
    uu _ z$est[,ec]+u
    u _ z$est[,ec]-u
    if (missing(ylim)) ylim _ range(c(u,uu))
  }
  else if (missing(ylim)) ylim _ range(z$est[,ec])
  if (plotopt) {
```

```
    if (missing(xlab)) xlab _ "Years"
    if (missing(ylab)) ylab _ "Log Hazard Ratio"
    if (!exists(".Device")) stop("Graphics device must be active")
    plot(z$est[,1],z$est[,ec],main=main,xlab=xlab,ylab=ylab,ylim=ylim,
      type="l",font=font,lwd=lwd,...)
    if (pvar) {
      lines(z$est[,1],uu,lty=2,lwd=lwd)
      lines(z$est[,1],u,lty=2,lwd=lwd)
    }
  }
  ylim
}
else if (z$type == "a") {                # plots for covs in additive model
  nsplinecov _ z$nvar[2]
# nsplinecov is the # spline covs in the model, ncovplot the # of the
# one being plotted
  o _ order(z$est[,ncovplot])
  ec _ nsplinecov+2*ncovplot-1
  if (pvar) {
    u _ ifelse(z$est[,ec+1]<0,0,z$est[,ec+1])
    u _ 2*sqrt(u)
    uu _ z$est[,ec]+u
    u _ z$est[,ec]-u
    if (missing(ylim)) ylim _ range(c(u,uu))
  }
  else if (missing(ylim)) ylim _ range(z$est[,ec])
  if (plotopt) {
    if (!exists(".Device")) stop("Graphics device must be active")
    if (missing(xlab)) xlab _ dimnames(z$est)[[2]][ncovplot]
    if (missing(ylab)) ylab _ "Log Hazard Ratio"
    plot(z$est[o,ncovplot],z$est[o,ec],main=main,xlab=xlab,ylab=ylab,ylim=ylim,
      type="l",font=font,lwd=lwd,...)
    if (pvar) {
      lines(z$est[o,ncovplot],uu[o],lty=2,lwd=lwd)
      lines(z$est[o,ncovplot],u[o],lty=2,lwd=lwd)
    }
    abline(h=0)
    if (knots) {
      ht _ .02*(ylim[2]-ylim[1])
      tk _ z$knots[4:(length(z$knots)-3)]
      ht_rep(ht,length(tk))
      segments(tk,ht,tk,-ht)
    }
  }
  ylim
}
}

cox.spline _ function(model.type,time,status,spline.cov,linear.cov,
strata,df,nknot,spline.knot,smooth.opt=1,smooth.param,output.opt="tests",
nest,maxiter=30,eps=1.e-4,rescale=T,ord=0,nstimes,stimes)  {

# nov:  parameters in call to fortran routines
#   1=no, 2=nrx(=no), 3=ncx, 4=istr (col # of x for strata, -1 if none,
#   5=maxiter, 6=knot optiont (1 use knots provided, 0 program calculates),
#   7=smoothing option (<0 use input smoothing params, 0 calc smoothing
#   param only in first iteration, >0 recalc after each iteration)
```

```
#   8=nfx (nfx+nsx in sph, nfx is #col in linear.cov, nsx in spline.cov),
#   9=nsx, 10=analysis option (<=0 estimates only, 1 est & var, >1 est,
#   var & test),
#     sph:  12=nknot, 13=nest
#     sphi:  11,12=nknx,nkny, 13,14=nestx,nesty, 15=rescale opt (0=yes)
#     stvc:  12=order of spline, 13=nest, 14=nknot, 15=nstimes

nov _ rep(0,15)
nov[1] _ length(time)
if (length(status) != nov[1]) stop("status wrong length")

nov[5] _ maxiter

if (missing(linear.cov)) {
  linear.cov _ as.matrix(spline.cov)
  namspl _ dimnames(linear.cov)[[2]]
  namlin _ NULL
  if (nov[1] != nrow(linear.cov)) stop("nrow mismatch in spline.cov")
  nov[9] _ ncol(linear.cov)
  nov[8] _ 0
}
else {
  linear.cov _ as.matrix(linear.cov)
  namlin _ dimnames(linear.cov)[[2]]
  if (nov[1] != nrow(linear.cov)) stop("nrow mismatch in linear.cov")
  nov[8] _ ncol(linear.cov)
  spline.cov _ as.matrix(spline.cov)
  if (nov[1] != nrow(spline.cov)) stop("nrow mismatch in spline.cov")
  nov[9] _ ncol(spline.cov)
  linear.cov _ cbind(linear.cov,spline.cov)
  namspl _ dimnames(spline.cov)[[2]]
}
if (length(namspl) != nov[9] & nov[9]>0)
          namspl_paste("spl",(1:nov[9]),sep="")
if (length(namlin) != nov[8] & nov[8]>0) namlin_paste("lin",(1:nov[8]),sep="")

if (missing(strata)) {
  nov[4] _ -1
}
else {
  if (length(strata) != nov[1]) stop("strata vector is the wrong length")
  strata _ as.factor(strata)
  linear.cov _ cbind(linear.cov,codes(strata))
  nov[4] _ ncol(linear.cov)
}
nov[3] _ ncol(linear.cov)

nomiss _  !(is.na(time) | is.na(status) |
            is.na(linear.cov %*% rep(1,nov[3])))
# check status variable
  if (any(status[nomiss] != 0 & status[nomiss] != 1))
          stop("invalid status values")
if (nov[4]<1) {
  status.table _ table(status[nomiss])
}
else {
  status.table _ table(strata[nomiss],status[nomiss])
```

```
}

nov[1] _ length(time[nomiss])
nov[2] _ nov[1]

if (nov[4] > 0) nstr _ length(unique(linear.cov[nomiss,nov[4]]))
else nstr _ 1

dsub _ rep(0,3*nov[9])
if (nov[9]>0) {
  if (!missing(smooth.param)) {
   if (model.type == "i") dsub[1] _ smooth.param
   else dsub[(nov[9]*2+1):(nov[9]*3)] _ smooth.param
   nov[7] _ -1
   }
  else {
   if (missing(df)) {
    if (model.type == "i") dsub[2] _ 8
    else dsub[1:nov[9]] _ rep(3,nov[9])
   }
   else {
    if (model.type == "i") {
       if (length(df) != 1) cat("warning:  length of df wrong\n")
       dsub[2] _ df
    } else {
      if (length(df) != nov[9]) cat("warning:  length of df wrong\n")
      dsub[1:nov[9]] _ df
    }
   }
   nov[7] _ max(smooth.opt,0)
  }
}

if (output.opt == "est") nov[10]_0
else if (output.opt == "var") nov[10]_1
else nov[10]_2

if (model.type == "a") {
  nfxx _ nov[8]
  nov[8] _ nov[8]+nov[9]
  linear.cov _ c(linear.cov[nomiss,],rep(0,nov[1]*nov[9]*4))

  if (missing(nknot)) nov[12] _ 10
  else nov[12] _ nknot[1]
  nknot _ nov[12]
  if (missing(spline.knot)) {
    nov[6] _ 0
    spline.knot _ rep(0,(nknot+6)*max(1,nov[9]))
  }
  else {
    if (length(spline.knot) != (nknot+6)*nov[9]) {
      nov[6] _ 0
      spline.knot _ rep(0,(nknot+6)*max(1,nov[9]))
    }
    else {
      nov[6] _ 1
    }
```

```
  }

  if (missing(nest)) nov[13] _ min(nov[1],100)
  else nov[13] _ min(nov[1],nest)

  np _ nov[8]+(nknot+2)*nov[9]
  iwork _ rep(0,3*nov[1]+max(np,nov[1])+1+4*nstr)
  isub _ rep(0,6+nov[9]+nov[9]*nov[1])
  if (nov[9]>0) for (i in 1:nov[8]) isub[6+i] _ i+nfxx
  nk3 _ nknot+3
  if (smooth.opt>0) {
    nk2 _ nknot+2
    nkk _ max(np-nk2,nk2)
    kmd _ max((np-nk2)*(np-nk2),nk3*nk3+2*nk2*nkk)
  }
  else {
    kmd _ nk3*nk3
  }
  ldsub _ 4*(nknot+4)*nov[9]+np*np+kmd+max(nov[1],4*np+np*np)
  ldsub _ max(ldsub,np*np+3*nk3*nk3)
  dsub _ c(dsub,rep(0,ldsub))
  u1 _ double(np)
  u2 _ double(1)
  v2 _ double(np*np)
#************************
  z _ double(nov[1]*np)
  z11 _ double(nov[1]*np)
  z22 _ double(nov[1]*np)
  gn _ integer(nov[3]*3)
  exz _ double(nov[1])
  s0 _ double(nov[1])
  s1 _ double(nov[1]*np)
  ts1 _ double(np)
  w _ double(nov[1]*np)
  wp _ double(nov[1]*np)
  ts1p _ double(np)
  id _ integer(nov[1])
  s1p _ double(np*nov[1])
  nf _ sum(status, na.rm=T)
#*****  print(nf)
#************************
  #cat('Befor sph\n')
  dsub _ .Fortran("sph",as.integer(nov),
       as.double(time[nomiss]),
       as.double(status[nomiss]),
       as.double(linear.cov),
       as.double(spline.knot),
       u1,u2,c(u2,u2),u1,
       as.double(v2),
       as.integer(iwork),
       as.double(dsub),
       as.integer(isub),
       double(3*2*nov[9]),
       double(nk3*2*nov[9]),
       as.double(eps),
       as.double(z),
       as.double(exz),
```

```
        as.double(s0),
        as.double(s1),
        as.double(ts1),
        as.double(w),
        as.double(wp),
        as.double(ts1p),
        as.integer(id),
        as.double(s1p),
        as.double(z11),
        as.double(z22),
        as.integer(gn))
#        as.integer(nf))
  #cat('After sph\n')

# print (dsub[[22]])


  if (dsub[[10]][1] <= 0) {
     if (dsub[[10]][1] < -2.5)
          stop("Possible overflow problems (No convergence)")
     else if (dsub[[10]][1] < -1.5)
          stop("Singular Penalized Information Matrix")
     else if (dsub[[10]][1] < -0.5) stop("Iteration did not converge")
     else stop("????")
  }

# things in output vector:
# lik, beta, var, inv 2nd deriv, knots, tests, eigenvalues, estimates
# dsub is dsub[[12]]
# 2nd deriv mat in dsub(3*nov[9]+1)-dsub(3*nov[9]+np*np)

if (nknot != dsub[[1]][12]) {
  nknot <- dsub[[1]][12]
  np <- dsub[[1]][8]+(nknot+2)*dsub[[1]][9]
  dsub[[10]] <- dsub[[10]][1:(np*np)]
  dsub[[6]] <- dsub[[6]][1:np]
}
 nsx _ dsub[[1]][9]
 #cat('nsx is:\n')
##############################################
# unpenalized information matrix
  indunp _ dsub[[13]][5]
  maxind _ indunp+np*np-1
  unpinf _ dsub[[12]][indunp:maxind]
  dim(unpinf) _ c(np,np)
##############################################
  penmind _ dsub[[13]][4]
#  penmaxi _ penmind + np*np-1
  penmaxi _ penmind + 4*(nknot+4)*nsx-1
#  penmaxi _ penmind + 4*nk3*nsx
#  penmatrix _ dsub[[12]][penmind:penmaxi]
  penm _ dsub[[12]][penmind:(indunp-1)]
#***** cat('isub3-isub5 are')
#*****  print(c(dsub[[13]][3],penmind,dsub[[13]][5]))
#*****  cat('4*(nknot+4)*nsx\n')
#*****  print(4*(nknot+4)*nsx)
#*****  cat('length of pen matrix\n')
```

```
#*****  print(indunp-penmind)
  dim(penm) _ c(nknot+4,4,nsx)
#*****  cat('************penality matrix***********\n')
#*****  print(penm)
#*****  cat('****************************************\n')
##############################################
#***** print(nsx)
#***** print(c((2*nsx+1):(3*nsx)))
#***** print(np)
 smoothp _ dsub[[12]][c((2*nsx+1):(3*nsx))]
#*****  cat('************smoothing parameters***********\n')
#*****  print(smoothp)
#*****  cat('dim of dsub[[12]]\n')
#*****  print(length(dsub[[12]]))
#*****  cat('for dft,dfa,alpha\n')
#*****  print(3*nsx)
#*****  cat('penality matrix\n')
#*****  print(4*(nknot+4)*nsx)
#*****  cat('unpen inf\n')
#*****  print(np*np)
#  cat('*************   DSUB[[12]]  ****************\n')
#  print(dsub[[12]])

#*****  cat('****************************************\n')
##############################################

nvar _ c(nfxx,nov[9])
names(nvar) _ c("linear","spline")
dim(dsub[[2]]) _ c(nov[1])
dim(dsub[[3]]) _ c(nov[1])
dim(dsub[[17]]) _ c(nov[1],np)
dim(dsub[[18]]) _ c(nov[1])
dim(dsub[[19]]) _ c(nov[1])
dim(dsub[[20]]) _ c(nov[1],np)
dim(dsub[[22]]) _ c(nov[1],np)
dim(dsub[[23]]) _ c(nov[1],np)
dim(dsub[[27]]) _ c(nov[1],np)
dim(dsub[[28]]) _ c(nov[1],np)
dim(dsub[[29]]) _ c(nov[3],3)
#print(dsub[[17]][1:10,])
#cat("cens indicator:\n")
#print(dsub[[3]][1:20])
#print(dsub[[2]][1:10])

#cat("z  :\n")
#print(dsub[[17]][1:20,])
#cat("z11 :\n")
#print(dsub[[27]][1:20,])
#cat("\n z22 :\n")
#print(dsub[[28]][1:20,])
#cat("\n gn :\n")
#print(dsub[[29]])
dim(dsub[[10]])_c(np,np)
####print(dsub[[10]])
#tempval _ as.data.frame(c(dsub[[2]],dsub[[3]],dsub[[17]][,c(1,2)]),
#     ncol=4)
#cat("event time, status, and covariate values\n")
```

```
#print(tempval)
dim(dsub[[17]]) _ c(nov[1],np)
#cat("covariate values\n")
#print(dsub[[17]][,c(1,2)])
#cat("event time and status\n")
#print(c(dsub[[2]],dsub[[3]]))
names(dsub[[6]]) _ c(namlin,namspl,rep(namspl,rep(nknot+2,nov[9])))
  dsub[[4]]_matrix(dsub[[4]],nrow=nov[1])[1:nov[13],c((nfxx+1):(nfxx+nov[9]),
  (nov[3]+1):(nov[3]+2*nov[9]))]
  v2_dsub[[12]][(3*nov[9]+1):(3*nov[9]+np*np)]
  dim(v2)_c(np,np)
  dim(dsub[[14]])_c(3,2*nov[9])
  dim(dsub[[15]])_c(nk3,2*nov[9])
  dsub[[5]]_matrix(dsub[[5]],ncol=max(nov[9],1))
  dimnames(dsub[[14]])_list(c("stat","pv","df"),rep(c("overall","linear"),
        nov[9]))
  dimnames(dsub[[4]])_list(NULL,c(namspl,rep(c("est","var"),nov[9])))
  dimnames(dsub[[15]])_list(NULL,rep(c("overall","lin"),nov[9]))
  if (output.opt=="est") {
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],
    est=dsub[[4]],knots=dsub[[5]],nknot = nknot,
    smooth.param=dsub[[12]][(nov[9]*2+1):(nov[9]*3)],table=status.table,
    type="a",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
    isx=dsub[[14]], infom=unpinf,
    z=dsub[[17]], exz=dsub[[18]], s0=dsub[[19]], s1=dsub[[20]],
    tent = dsub[[2]], csin = dsub[[3]], penm = penm) }
  else if (output.opt=="var") {
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],var=dsub[[10]],d2i=v2,
    est=dsub[[4]],knots=dsub[[5]],nknot = nknot,
    smooth.param=dsub[[12]][(nov[9]*2+1):(nov[9]*3)],table=status.table,
    type="a",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
    isx=dsub[[14]], infom=unpinf,
    z=dsub[[17]], exz=dsub[[18]], s0=dsub[[19]], s1=dsub[[20]],
    tent = dsub[[2]], csin = dsub[[3]], penm = penm) }
  else {
    test _ vector("list",nov[9])
    names(test) _ namspl
    for (i in 1:nov[9]) test[[i]] _ dsub[[14]][,c(2*i-1,2*i)]
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],var=dsub[[10]],d2i=v2,
    est=dsub[[4]],knots=dsub[[5]],tests=test,eigs=dsub[[15]],nknot=nknot,
    smooth.param=dsub[[12]][(nov[9]*2+1):(nov[9]*3)],table=status.table,
    type="a",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
    isx=dsub[[14]], infom=unpinf,
    z=dsub[[17]], exz=dsub[[18]], s0=dsub[[19]], s1=dsub[[20]],
    tent = dsub[[2]], csin = dsub[[3]], penm = penm) }
}
else if(model.type == "i") {
  if (ncol(spline.cov) != 2)
     stop("Only 2 covariates allowed for tensor product spline")

linear.cov _ c(linear.cov[nomiss,],rep(0,nov[1]*8))

if (missing(nknot)) nov[11:12] _ 3
else nov[11:12] _ nknot
if (missing(spline.knot)) {
  nov[6] _ 0
  spline.knot _ rep(0,nov[11]+nov[12]+12)
```

```
}
else {
  if (length(spline.knot) != (nov[11]+nov[12]+12)) {
    nov[6] _ 0
    spline.knot _ rep(0,nov[11]+nov[12]+12)
  }
  else {
    nov[6] _ 1
  }
}

if (missing(nest)) nov[13:14] _ 20
else nov[13:14] _ nest

nb _ (nov[11]+4)*(nov[12]+4)-1
np _ nov[8]+nb
iwork _ rep(0,3*nov[1]+max(np,nov[1])+1+4*nstr)
isub _ rep(0,6+nov[8]+2*nov[1])
for (i in 1:nov[8]) isub[6+i] _ i
isub[1] _ nov[8]+1
isub[2] _ nov[8]+2
ldsub _ max(nov[13]*nov[14]*4+np*np,16*(nov[11]+4)*(nov[12]+4)+nb*nb+np*np+
   max(max(nov[1],4*np+np*np),np*np+nb*nb))
dsub _ c(dsub[1:3],rep(0,ldsub))

if (rescale) nov[15] _ 0
else nov[15] _ 1

u1 _ double(np)
u2 _ double(1)
v2 _ double(np*np)
#************************
  z _ double(nov[1]*np)
 z1 _ double(nov[1]*np)
  exz _ double(nov[1])
  s0 _ double(nov[1])
  s1 _ double(nov[1]*np)
  ts1 _ double(np)
  w _ double(nov[1]*np)
  wp _ double(nov[1]*np)
  ts1p _ double(np)
  id _ integer(nov[1])
  s1p _ double(nov[1]*np)
  nf _ sum(status, na.rm=T)
#************************

dsub _ .Fortran("sphi",as.integer(nov),
       as.double(time),
       as.double(status),
       as.double(linear.cov),
       as.double(spline.knot),
       u1,u2,c(u2,u2),u1,
       as.double(v2),
       as.integer(iwork),
       as.double(dsub),
       as.integer(isub),
       double(12),
```

```
        double(4*nb),
        as.double(eps),
        as.double(z),
        as.double(exz),
        as.double(s0),
        as.double(s1),
        as.double(ts1),
        as.double(w),
        as.double(wp),
        as.double(ts1p),
        as.integer(id),
        as.double(s1p),
        as.integer(z1))
#        as.integer(nf))

  if (dsub[[10]][1] <= 0) {
     if (dsub[[10]][1] < -2.5)
         stop("Possible overflow problems (No convergence)")
     else if (dsub[[10]][1] < -1.5)
         stop("Singular Penalized Information Matrix")
     else if (dsub[[10]][1] < -0.5) stop("Iteration did not converge")
     else stop("????")
  }

# things in output vector:
# lik, beta, var, inv 2nd deriv, knots, tests, eigenvalues, estimates
# dsub is dsub[[12]]
# estimates in dsub(4)-dsub(3+nestx*nesty*4)
# 2nd deriv mat in dsub(4+nestx*nesty)-dsub(3+nestx*nesty+np*np)

  nvar _ nov[8:9]
  names(nvar) _ c("linear","spline")

  names(dsub[[6]])_c(namlin,paste("spline",1:nb,sep=""))
  ne2 _ nov[13]*nov[14]
  linear.cov _ dsub[[12]][4:(3+ne2*4)]
  dim(linear.cov) _ c(ne2,4)
  v2 _ dsub[[12]][(4+4*ne2):(3+4*ne2+np*np)]
  dim(v2) _ c(np,np)
  dim(dsub[[10]]) _ c(np,np)
  dim(dsub[[14]]) _ c(3,4)
  dim(dsub[[15]]) _ c(nb,4)
  dimnames(dsub[[14]]) _ list(c("stat","pv","df"),
       c("overall",namspl,"interact"))
  dimnames(linear.cov) _ list(NULL,c(namspl,"est","var"))
  dimnames(dsub[[15]]) _ list(NULL,c("overall",namspl,"interact"))
  if (output.opt=="est") {
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],
      est=linear.cov,nknot=nknot,
      knots=dsub[[5]], smooth.param=dsub[[12]][1],table=status.table,
      type="i",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
      isx=dsub[[14]], infom=unpinf)
  }
  else if (output.opt=="var") {
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],var=dsub[[10]],
      d2i=v2,est=linear.cov,nknot=nknot,
      knots=dsub[[5]],smooth.param=dsub[[12]][1],table=status.table,
```

```
      type="i",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
      isx=dsub[[14]], infom=unpinf)
  }
  else {
    list(loglik=c(dsub[[7]],dsub[[8]]),coef=dsub[[6]],var=dsub[[10]],
      d2i=v2,est=linear.cov,knots=dsub[[5]],nknot=nknot,
      tests=dsub[[14]],eigs=dsub[[15]],smooth.param=dsub[[12]][1],
      table=status.table,type="i",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
      isx=dsub[[14]],infom=unpinf)
  }
}
else if (model.type == "t") {

if (ord != 0 & ord != 2 & ord != 3) stop("ord must be 0,2 or 3 for tvc")

nov[12] _ ord

if (missing(nknot)) nknot _ 10
  else nknot _ nknot[1]
nov[14] _ nknot
if (missing(spline.knot)) {
  nov[6] _ 0
  spline.knot _ rep(0,(nknot+6))
}
else {
  if (length(spline.knot) != (nknot+6)) {
    nov[6] _ 0
    spline.knot _ rep(0,(nknot+6))
  }
  else {
    nov[6] _ 1
  }
}
iord1 _ ord+1
ncp _ iord1
if (missing(nest)) nest_100
if (ord==0) {
  nstimes _ nknot
# if ord==0 use interior knots for switch times
# so if knots specified need to use them.
  if (nov[6]==1) stimes _ spline.knot[4:(nknot+3)]
  else stimes _ rep(0,nknot)
  ntud _ nstimes+1
  ncp _ 2
  nww _ ntud
  nest _ 2*ntud
}
else {
 if (!missing(stimes)) {
   nstimes _ length(stimes)
   ntud _ nstimes+1
   nww _ ntud
   nest _ 2*ntud
 }
 else {
   if (missing(nstimes)) {
     nstimes _ nov[1]+1
```

```
      stimes _ 0
      ntud _ 1
      nww _ nov[1]
    }
    else {
      stimes _ rep(0,nstimes)
      ntud _ nstimes+1
      nww _ ntud
      nest _ 2*ntud
    }
  }
}
nov[15] _ nstimes
nov[13] _ nest
nbas _ nknot+iord1
np _ nov[8]+nov[9]*nbas
iwork _ rep(0,3*nov[1]+max(np,nov[1])+1+(2+2*ntud)*nstr)
isub _ rep(0,8+nov[8]+nov[9]+nww)
l _ 0
if (nov[8]>0) for (i in 1:nov[8]) {
  l _ l+1
  isub[8+l] _ 1
}
for (i in 1:nov[9]) {
  l _ l+1
  isub[8+l] _ 1
}
if (smooth.opt>0) {
  nkk _ max(np-nbas,nbas)
  kmd _ max((np-nbas)*(np-nbas),nbas*nbas+2*nbas*nkk)
}
else {
  kmd _ nbas*nbas
}
ldsub _ ncp*(nknot+4)+iord1*nww+np*np+kmd+max(nov[1],4*np+np*np)
ldsub _ max(ldsub,iord1*nww+nest*(2*nov[9]+1))
dsub _ c(dsub,rep(0,ldsub))
u1 _ double(np)
u2 _ double(1)
v2 _ double(np*np)
#************************
  z _ double(nov[1]*np)
 z1 _ double(nov[1]*np)
  exz _ double(nov[1])
  s0 _ double(nov[1])
  s1 _ double(nov[1]*np)
  ts1 _ double(np)
  w _ double(nov[1]*np)
  wp _ double(nov[1]*np)
  ts1p _ double(np)
  id _ integer(nov[1])
  s1p _ double(nov[1]*np)
  nf _ sum(status, na.rm=T)
#************************

dsub _ .Fortran("stvc",as.integer(nov),
       as.double(time[nomiss]),
```

```
        as.double(status[nomiss]),
        as.double(linear.cov[nomiss,]),
        as.double(spline.knot),
        as.double(stimes),
        u1,u2,c(u2,u2),u1,
        as.double(v2),
        as.integer(iwork),
        as.double(dsub),
        as.integer(isub),
        as.double(3*2*nov[9]),
        as.double(nbas*2*nov[9]),
        as.double(v2),
        as.double(eps),
        as.double(z),
        as.double(exz),
        as.double(s0),
        as.double(s1),
        as.double(ts1),
        as.double(w),
        as.double(wp),
        as.double(ts1p),
        as.integer(id),
        as.double(s1p),
        as.integer(z1))
#        as.integer(nf))

  if (dsub[[11]][1] <= 0) {
     if (dsub[[11]][1] < -2.5)
          stop("Possible overflow problems (No convergence)")
     else if (dsub[[11]][1] < -1.5)
          stop("Singular Penalized Information Matrix")
     else if (dsub[[11]][1] < -0.5) stop("Iteration did not converge")
     else stop("????")
  }

# things in output vector:
# lik, beta, var, inv 2nd deriv, knots, tests, eigenvalues, estimates
# dsub is dsub[[13]]

  nvar _ nov[8:9]
  names(nvar) _ c("linear","spline")

names(dsub[[7]]) _ c(namlin,rep(namspl,rep(nbas,nov[9])))
dim(dsub[[11]]) _ c(np,np)
dim(dsub[[17]]) _ c(np,np)
j1 _ dsub[[14]][5]
if (ord==0) nest_2*dsub[[1]][14]+2
linear.cov _ dsub[[13]][j1:(j1-1+nest*(2*nov[9]+1))]
dim(linear.cov) _ c(nest,2*nov[9]+1)
dim(dsub[[15]]) _ c(3,2*nov[9])
dim(dsub[[16]]) _ c(nbas,2*nov[9])
dimnames(dsub[[15]]) _ list(c("stat","pv","df"),
  rep(c("overall","nonprop"),nov[9]))
dimnames(linear.cov) _ list(NULL,c("times",rep(c("est","var"),nov[9])))
dimnames(linear.cov)[[2]][2*(1:nov[9])] _ namspl
dimnames(dsub[[16]]) _ list(NULL,rep(c("overall","nonprop"),nov[9]))
```

```
test _ vector("list",nov[9])
names(test) _ namspl
for (i in 1:nov[9]) test[[i]] _ dsub[[15]][,c(2*i-1,2*i)]

if (output.opt=="est") {
list(loglik=c(dsub[[8]],dsub[[9]]),coef=dsub[[7]],
  est=linear.cov,knots=dsub[[5]],nknot=nknot,
  smooth.param=dsub[[13]][(nov[9]*2+1):(nov[9]*3)],stimes=dsub[[6]],
  table=status.table,type="t",nvar=nvar,w=dsub[[22]],wp=dsub[[23]],
  isx=dsub[[14]],infom=unpinf)
}
else if (output.opt=="var") {
list(loglik=c(dsub[[8]],dsub[[9]]),coef=dsub[[7]],var=dsub[[11]],d2i=dsub[[17]],
  est=linear.cov,knots=dsub[[5]],nknot=nknot,
  smooth.param=dsub[[13]][(nov[9]*2+1):(nov[9]*3)],stimes=dsub[[6]],
  table=status.table,type="t",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
  isx=dsub[[14]],infom=unpinf)
}
else {
list(loglik=c(dsub[[8]],dsub[[9]]),coef=dsub[[7]],var=dsub[[11]],d2i=dsub[[17]],
  est=linear.cov,knots=dsub[[5]],tests=test,eigs=dsub[[16]],nknot=nknot,
  smooth.param=dsub[[13]][(nov[9]*2+1):(nov[9]*3)],stimes=dsub[[6]],
  table=status.table,type="t",nvar=nvar,w=dsub[[22]], wp = dsub[[23]],
  isx=dsub[[14]], infom=unpinf)
}
}
}

#jasa.data <- read.table("jasa.data",header=T,row.names="id")
```

APPENDIX 5. COPY OF FORTRAN CODE NEEDED TO RUN GRAY'S MULTIPLE OUTCOME SURVIVAL MODEL

```
c
c This software comes with absolutely no guarantees.  You have permission to
c use it for any noncommercial purpose, and to modify it as needed.
c Copyright 1992 by Robert Gray
c
      subroutine cests(x,no,nfx,nsx,beta,np,nknot,tknot,nkl,elp)
c routine for calculating function estimates and variances
c from output of sph/wsph.
c x is covariate values including spline terms.  for info on ordering
c etc see sph.f
      double precision x(no,nfx),w(4),elp(no)
      double precision beta(np),tknot(-2:(nkl+3),nsx)
      isx=nfx-nsx
      nk2=nknot+2
      do 88 i=1,no
        elp(i)=0
        do 10 j=1,nfx
 10     elp(i)=elp(i)+x(i,j)*beta(j)
        do 5 k=1,nsx
          call spln(nknot,3,tknot(-2,k),x(i,isx+k),1,intbl,w,1)
          l3=nfx+nk2*(k-1)+intbl-1
          l4=min(nk2,intbl+3)-intbl+1
          do 6 iq=1,l4
 6        elp(i)=elp(i)+w(iq)*beta(l3+iq)
 5      continue
        elp(i)=exp(elp(i))
 88   continue
      return
      end

      subroutine mresd(s,c,x,n,nstr,nostr)
      double precision s(n),c(n),x(n),xb,dd,ch
      integer nostr(nstr)
c
c  subroutine to calculate martingale residuals in proportional
c  hazards models
c                                      and s the survival times
c  on input c is the censoring indicator ^ and x is exp(x'b) for each case
c  on output x is overwritten with resids for each case.
c  data is assumed sorted on survival (ascending) within strata
c
      l1=1
      do 10 k=1,nstr
        xb=0
        do 19 i=l1,nostr(k)
 19     xb=xb+x(i)
        ch=0
        dd=0
        l3=l1
        do 20 i=l1,nostr(k)
          if (s(i).eq.s(l3)) then
            dd=dd+c(i)
          else
            if (dd.gt.0) ch=ch+dd/xb
            dd=c(i)
            do 22 ii=l3,i-1
              xb=xb-x(ii)
              x(ii)=c(ii)-x(ii)*ch
 22           continue
              l3=i
          endif
 20     continue
        if (dd.gt.0) ch=ch+dd/xb
```

```
      do 23 ii=l3,nostr(k)
   23 x(ii)=c(ii)-x(ii)*ch
      ll=nostr(k)+1
   10 continue
      return
      end

      subroutine calvar(vv,np,lvv,v3,lv3,var,lvar,work)
c input: vv actual 2nd deriv matrix, v3 var cov matrix of the scores
c np matrices are npxnp, with actual row dimensions given by l* terms
c output: var is var cov matrix, vv^-1 v3 vv^-1. vv and var can be the
c same.  work (entries 1 to np*np) will have vv^-1
      double precision vv(lvv,np),v3(lv3,np),var(lvar,np),work(np,np)
c  first compute full variance covariance matrix:
      call pdi(np,vv,work,lvv,np)
      do 10 i=1,np
      do 10 j=i,np
      var(i,j)=0
      do 11 ii=1,np
      do 11 jj=1,np
   11 var(i,j)=var(i,j)+work(i,ii)*v3(ii,jj)*work(jj,j)
      var(j,i)=var(i,j)
   10 continue
      return
      end

      subroutine coxrg(s,c,x,n,nrx,ncx,istrat,np,mit,iflg,ntime,
     &times,beta,lik0,lik,scv,vinf,iwork,dwork,act,eval,sub2,
     &isub,dsub,eps,icnv,z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1,gn)
c input:
c        s   survival times
c        c   censoring indicator (1=failure, 0=censoring)
c        x   matrix, ith row contains covariate information on
c            the ith  case.
c        n   sample size (# rows used in s,c and x)
c    nrx,ncx actual row dim of x in calling program, and number of
c            cols of x actually used
c    istrat  column of x which contains the strata variable.  istrat<=0
c            means only one stratum.
c        np  total # parameters in the model
c        mit max # iterations, included primarily so you
c          can specify 1 to get info for score tests.
c       iflg this program creates a variety of indexing information
c            which is stored in portions of iwork.
c            and also does some permuting of rows of s,c,and
c            x.  On subsequent calls with the same values of
c            s,c,x, can save recalculating this by setting
c            iflg>0.  If iflg<=0 then this info is calculated
c      ntime # of times at which time varying covariates (tvc) are allowed
c            to change values.  If ntime=0 assumes no tvc.  Set ntime to
c            a value>n to use all failure times.
c      times a vector of length ntime containing the times where
c            covariates are allowed to change values.  If ntime>n or
c            ntime<=0 then times is not used.  If times(ntime)<0 then
c            ntime timepoints will be generated and stored in times,
c            with approx equal #'s of failures between the times.
c       beta initial value of parameters.  Usually should set this
c            to 0 before the call.  Can be set to other values to
c            generate info for score tests, etc.
c       act,eval,sub2  names of user supplied subroutines called by this
c            program (see below)
c       isub,dsub  integer and double precision arrays containing
c            information that is passed to the subroutines act,eval
```

```
c            & sub2
c       eps  convergence criteria, stops when change in loglikelihood
c            is < eps
c
c  output:
c       on output rows of s c and x may have been permuted
c       beta values of paramaters after termination of iteration
c       lik0 value of likelihood at initial parameter values
c       lik  value of log likelihood after final iteration
c       scv  value of the score vector components after termination
c       vinf information matrix after termination
c       icnv  0=converge, 1=did not converge, 2=sing inf matrix
c             3=could not find a better point, or overflow probs
c
c  workspace:
c      iwork integer, of length at least 3*n+max(n,np)+1+2*nstr+2*ntud*nstr,
c        where ntud is ntime+1,or ntud=1 (when ntime>n)
c      dwork double precision, of lenght at least max(n,4*np+np*np)
c
c  subroutines:
c   act, eval, and sub2 are the names of subroutines that must be
c   provided by the user.  The names of these routine must be declared
c   external in the calling program.  act is a routine that specifies
c   which model terms are active in the current time interval.  The
c   reason for including this is that if your model includes tvc that
c   are products of fixed covariates with functions of time (created to
c   give flexible hazard ratio models over time), then if the functions
c   of time have local support properties (ie are 0 outside of a fixed
c   interval) then considerable efficiency can be gained by indicating
c   this through act.  The call is
c      call act(lci,l1,l2,idx,n,sv,ce,xx,nrx,ncx,iact,nact,np,
c     &lc2,isub,dsub)
c   where lci indexes the time interval, and lc2 the strata.  l1 and l2
c   and idx are such that sv(idx(l1,1)) to sv(idx(l1,2)) are the
c   survival times of the subjects who fail at the smallest failure time
c   in the interval, or are censred between this time and the next.  l2
c   gives the same thing for the largest time.  sv are the survival
c   times, ce the censoring times, and xx the cov info, n the sample size,
c   iact as output will need to contain which of the np covariates are active
c   on this time interval, eg if nact=4 and iact=c(1,3,4,6), it would
c   mean that there are contributions from the 1st, third, fourth and sixth
c   covariates in this time interval, but that the score and information
c   contributions of the other covariates are identically 0.
c
c   eval is a routine which specifies the value of the covariate vector
c   for subject i at time sv(l).  The call is
c      call eval(g,gp,b,lci,i,l,sv,ce,xx,nrx,ncx,n,np,lc2,nact,iact,
c     &isub,dsub),
c   where gp(j) will be the value of the jth cov (only gp(iact(j)),j=1,
c   nact will actually be used).  lci gives the number of the time
c   interval, i the observation number, sv(l) the failure time of the
c   current likelihood contribution, with the other arguments as
c   specified in act.  This routine also needs to calculate g=x'b for
c   case i.
c
c   sub2 is a routine called after each iteration.  This can be a null
c   routine, or could be used to apply a penalty function to the scores
c   and information (for example).  The call is
c      call sub2(np,beta,lik,svc,vinf,vvi,isub,dsub,nit)
c   where nit is the current iteration count.  One additional call
c   is made after convergence, with nit=-1. (vvi is a working matrix
c   of size np*np, included because the extra space is sometimes needed)
c      change 4-18-91:  extra call after convergence no longer made.
```

```
c
c   In addition to the routines in this file (coxft, coxg, strat, ut,
c     uft, tint) and the user
c   supplied routines, calls are made to tint4 cholg, solve, dperm and sortg,
c   which are in ~gray/src/util
c
      double precision s(n),c(n),x(nrx,ncx),scv(np),lik0,lik,
     &vinf(np,np),dwork(1),times(1),beta(np),dsub(1),eps
C*************************************************************
      double precision z(n,np),exz(n),s0(n),s1(n,np),z1(n,np)
      double precision ts1(np),w(n,np)
C*************************************************************
C>>>>>
      double precision wp(n,np),ts1p(np),s1p(np,n)
      integer id(n),gn(1)
C<<<<<

      integer iwork(1),isub(1)
      external act,eval,sub2
C>>>>>>
C          do 15 i = 1, n
C            do 10 j = 1, np
C               z1(i,j) = x(i,j)
c  10        continue
c  15      continue
C<<<<<<
      mnstr=1
      miuti=mnstr+1
      mtduti=miuti+2*n
c miwk needs lenght n
      miwk=mtduti+n
      mnostr=miwk+max(n,np)
      mdwk=1
      if (ntime.lt.n) then
c ntud is the # time intervals
      ntud=ntime+1
      if (ntud.gt.1.and.times(ntime).le.0) then
          do 5 i=1,n
    5     iwork(miwk+i-1)=i
          call sortg(s,iwork(miwk),1,n)
          call dperm(c,iwork(miwk),1,n,dwork(mdwk))
        do 6 j=1,ncx
    6     call dperm(x(1,j),iwork(miwk),1,n,dwork(mdwk))
c          do 15 i = 1, n
c            do 10 j = 1, np
c               z1(i,j) = x(i,j)
c  10        continue
c  15       continue
        call tint4(ntud,s,c,times,n,iwork(miwk))
c       write (6,*) (times(j),j=1,ntime)
      endif
      else if (ntud.lt.1) then
      ntud=1
      endif
c index information:
      if (iflg.le.0) then
      call strat(istrat,s,c,x,iwork(mnostr),nstr,iwork(miwk),
     &dwork(mdwk),n,ncx,nrx,iwork(miuti),iwork(mtduti))
        iwork(mnstr)=nstr
      else
        nstr=iwork(mnstr)
      endif
c      write (6,*) nstr,n
```

```
c      write (6,*) (iwork(mnostr+i-1),i=1,nstr)
      mnuti=mnostr+nstr
c      write (6,*) (iwork(mnuti+i-1),i=1,nstr)
c      call pr1(iwork(miuti),iwork(mtduti),n)
      mitud=mnuti+nstr
c length of itud is ntud*nstr*2
      mdvi=mdwk+4*np
c time intervals:
      if (ntime.lt.n) then
        l1=1
        do 225 l1=1,nstr
      l2=(l1-1)*2*ntud
        call tint(ntud,times,s,c,iwork(mitud+l2),n,iwork(miuti),
     &  iwork(mnuti+l1-1),l1)
        l1=iwork(mnostr+l1-1)+1
 225    continue
      endif
CC>>>>>>
C          do 15 i = 1, n
C            do 10 j = 1, np
C               z1(i,j) = x(i,j)
C  10        continue
C  15      continue
CC<<<<<<
c      call pr2(iwork(mitud),ntud,nstr)
c fit model:
      call coxft(s,c,x,ncx,nrx,n,np,beta,lik0,lik,scv,vinf,mit,
     &iwork(miuti),iwork(mtduti),iwork(mnuti),iwork(mitud),ntud,
     &ntime,nstr,iwork(mnostr),iwork(miwk),dwork(mdvi),dwork(mdwk),
     &act,eval,sub2,isub,dsub,eps,icnv,z,exz,s0,s1,ts1,w,
     &wp,ts1p,id,s1p,z1,gn)
      return
      end

      subroutine coxft(s,c,x,ncx,nrx,n,np,beta,lik0,lik,sb,vv,mit,
     &iuti,tduti,nuti,itud,ntud,ntime,nstr,nostr,iwork,vvi,work,
     &act,eval,sub2,isub,dsub,eps,icnv,z,exz,s0,s1,ts1,w,
     &wp,ts1p,id,s1p,z1,gn)
      double precision s(n),c(n),x(nrx,ncx),beta(np),lik0,liko,lik
      double precision sb(np),vv(np,np),vvi(np,np),work(1),dsub(1)
      double precision eps,shrnk,eps2
C*************************************************************
      double precision z(n,np),exz(n),s0(n),s1(n,np),z1(n,np)
      double precision ts1(np),w(n,np)
C*************************************************************
C>>>>>
      double precision wp(n,np),ts1p(np),s1p(np,n)
      integer id(n),gn(ncx,3)
C<<<<<
      integer iuti(n,2),tduti(n),nuti(nstr),itud(ntud,2,nstr)
      integer iwork(1),isub(1),nostr(nstr)
      external act,eval
      do 10 i = 1, n
        do 5 j = 1, np
           w(i,j) = 0.d0
          wp(i,j) = 0.d0
    5 continue
   10 continue
      eps2=1.d+0
      shrnk=.5d0
      mshrk=4
      icnv=1
      np1=np+1
```

```
      call coxg(ntime,n,np,beta,s,c,x,ncx,nrx,iuti,itud,ntud,
     &tduti,nuti,sb,vv,lik,
     &vvi,work(2*np+1),work(3*np+1),nstr,nostr,iwork,
     &act,eval,isub,dsub,iprb,z,exz,s0,s1,ts1,w,id,gn,nfx)

C>>>>>>
         do 2 i = 1, n
           do 1 j = 1, np
              z1(i,j) = x(i,j)
  1        continue
  2      continue
C<<<<<<

c      write (6,100) ((vv(i,j),j=1,np),i=1,np)
c100   format (4e18.8)
      if (iprb.gt.0) then
           icnv=3
           return
      endif
      call sub2(np,beta,lik,sb,vv,vvi,isub,dsub,1)
      lik0=lik
      liko=lik
c      write (6,*) lik
      do 15 i=1,np
  15  work(i)=beta(i)
      call cholg(np,vv,vvi,np,np)
      if (vvi(1,1).lt.0) then
         icnv=2
         return
      endif
      call solve(np,vvi,sb,work(np1),np)
      do 58 i=1,np
  58  beta(i)=beta(i)+work(np+i)
      if (mit.le.1) return
      nshrk=0
      do 188 nit=2,mit
      call coxg(ntime,n,np,beta,s,c,x,ncx,nrx,iuti,itud,ntud,
     &tduti,nuti,sb,vv,lik,
     &vvi,work(2*np+1),work(3*np+1),nstr,nostr,iwork,
     &act,eval,isub,dsub,iprb,z,exz,s0,s1,ts1,w,id,gn,nfx)
      call sub2(np,beta,lik,sb,vv,vvi,isub,dsub,nit)
c      write (6,*) lik
      if (iprb.gt.0.or.lik.lt.liko-eps2) then
        if (nshrk.le.mshrk) then
          nshrk=nshrk+1
c         write (6,110)
 110      format('shrinking')
        do 56 i=1,np
          work(np+i)=shrnk*work(np+i)
  56      beta(i)=work(i)+work(np+i)
        go to 188
        else
          icnv=3
          return
        endif
c      else
c        nshrk=0
      endif
      do 455 i=1,np
 455  work(i)=beta(i)
      call cholg(np,vv,vvi,np,np)
      if (vvi(1,1).lt.0) then
         icnv=2
```

```
            return
          endif
          call solve(np,vvi,sb,work(np1),np)
          do 456 i=1,np
  456     beta(i)=beta(i)+work(np+i)
          if (abs(lik-liko).le.eps.and.nshrk.le.0) then
c           liko=0
c           do 668 i=1,np
c 668       liko=liko+work(np+i)*work(np+i)
c           if (liko.lt.np*1.d-6) then
             icnv=0
             go to 920
c           endif
          endif
          nshrk=0
          liko=lik
  188     continue
  920     continue

CCCCCCCCCCCCC     ************************************************
CC        call realpr('Covariate values')
          do 925 i = 1, n
            if(.NOT.(c(i).gt.0.d-10))then
              do ii=1, nfx
                z(i,ii) = z1(i,ii)
              enddo
            do ii=1, ncx
                l1 = gn(ii,1)
                l3 = gn(ii,2)
                l4 = gn(ii,3)
                do l = 1,l4
                   z(i,l3+l)=z1(i,l1+l)
                enddo
              enddo
            else
              goto 930
            endif
    925   continue

    930   ict = 0
          do kk=1,n
             ict=ict+c(kk)
          enddo
          do i = 1, n
            exz(i) = 0.d0
            do j = 1, np
              exz(i) = exz(i) + beta(j)*z(i,j)
              s1(i,j) = 0.d0
              s1p(j,i) = 0.d0
            enddo
CC          call realpr(z(i,1),1)
            exz(i) = dexp(exz(i))
            s0(i) = 0.d0
          enddo
          do i = n, 1, -1
             if(i.eq.n)then
               s0(i) = exz(i)
               do j = 1, np
                 s1(i,j)=z(i,j)*exz(i)
               s1p(j,i) = z(i,j)-s1(i,j)/s0(i)
               enddo
             else
               s0(i) = s0(i+1) + exz(i)
```

```
            do j = 1, np
              s1(i,j) = s1(i+1,j) + z(i,j)*exz(i)
              s1p(j,i) = z(i,j)-s1(i,j)/s0(i)
            enddo
          endif
          call sub8n(np,beta,s1p(1,i),isub,dsub,n)
        enddo
CCCCCC ******
        do 9 I = 1, n
          do 6 j = 1, np
            ts1(j) = 0.d0
            ts1p(j) = 0.d0
            mf = 0
            do 1 = 1, i
             mf = mf + c(1)
            enddo
            do 3 1 = 1, i
              if(c(1).gt.1.d-10) then
                ts1(j) = ts1(j) + (exz(i)/s0(1))*(z(i,j) -
     &                   s1(1,j)/s0(1))
C                ts1p(j) = ts1p(j)+(exz(i)/s0(1))*(z(i,j) +
C     &    (s1p(j,1)+s1(1,j)/s0(1) - z(1,j))*ict/mf-s1(1,j)/s0(1))
              ts1p(j)=ts1p(j)+(exz(i)/s0(1))*(z(i,j)-s1(1,j)/s0(1))
            endif
    3   continue
    6   continue
        if(c(i).gt.1.d-10)then
          do j=1, np
            w(i,j) = (z(i,j)-s1(i,j)/s0(i)) - ts1(j)
            wp(i,j) = s1p(i,j) - ts1p(j)
          enddo

        else
          do j = 1, np
            w(i,j) = -ts1(j)
            wp(i,j) = -ts1p(j)
          enddo
        endif

c         write(6,*)(w(i,11),11=1,26)
    9   continue
CCCCCCCCCCCCC      ************************************************
        return
        end

        subroutine sub8n(np2,beta,sb,isub,dsub,nf)
        double precision beta(np),sb(np),dsub(1)
        integer isub(1)
        common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
       &nk2,nk3,nk4,malph
        call sub9n(beta,sb,dsub(isub(4)), dsub(malph),nf)
        return
        end

        subroutine sub9n(beta,sb,penm,alpha,nf)
        double precision beta(np),sb(np)
        double precision penm(nk4,4,nsx),alpha(nsx)
        common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
       &nk2,nk3,nk4,malph
        do 535 j=1,nsx
        11=nfx+1+(j-1)*nk2
        call penscor(nk2,beta(11),penm(1,1,j),nk4,4,sb(11),alpha(j),
       &np,no,nf)
```

```
 535  continue
      return
      end

      subroutine penscor(nb,beta,penm,mr,md,s,alpha,np,n,nf)
c given a penalty matrix penm and parameter values beta,
c subtracts penalty terms from the scores
c the beginning index of beta and s (the score) in the
c call should correspond to where the spline terms begin
c ie call penlik(...beta(7)...s(7)...) if the spline is
c stored consecutively beginning with the 7th component
c nk is the number of knots, nb is the number
c of basis fcns actually in use, alpha is the value of the
c smoothing parameter. Aug. 6' 2001
      double precision penm(mr,md),beta(1),s(1),alpha,u
      do 10 i=1,nb
      u=0
      m1=max(1,i-md+1)
      m2=min(nb,i+md-1)
      l=0
      do 12 j=i,m2
      l=l+1
      u=u+penm(i,l)*beta(j)
  12  continue
      if (m1.lt.i) then
        l=i-m1+2
        do 11 j=m1,i-1
        l=l-1
        u=u+beta(j)*penm(j,l)
  11    continue
      endif
      s(i)=s(i)-(alpha*u/nf)
  10  continue
      return
      end

      subroutine coxg(nt,n,np,b,sv,ce,xx,ncx,nrx,idx,it,ntud,nd,
     &ndf,s,v,lik,vt,xb,gp,nstr,nostr,iact,act,eval,isub,dsub,iprb,
     &z,exz,s0,s1,ts1,w,id,gn,nfx)
c  subroutine to calculate cox likelihood, score, and inf
C  nt is # time intervals= #switch points +1.
c  n is the # of observations, np the # parameters, b the current
c parameter values, idx and it are index information:  idx should be
c from uft, giving for each strata the increment in the risk set
c at each unique failure time.  it should be from tint, and gives
c the rows of idx which correspond to failures in each of the time
c intervals.  nd is the # deaths at each unique failure time in each
c strata.  ndf is the # of unique failure times (nuti in other programs)
c in each strata
c On output, s v and lik are the score, information (-second deriv)
c and likelihood.  vt is an npxnp matrix xb and g are np dimensional
c vectors.
c also, iact is an integer working vector of length np.
c act and eval need to be names of subroutines that will return
c the active covariates within each time interval (act), and
c the covariate values for a specified obs within a specified interval
c (eval)
      double precision b(np),s(np),v(np,np),lik,u,dsub(1)
      double precision xb(np),xb1,xb1o,u1,u2,vt(np,np),g,gp(np)
      double precision sv(n),ce(n),xx(nrx,ncx)
C*************************************************************
      double precision z(n,np),exz(n),s0(n),s1(n,np)
      double precision ts1(np),w(n,np)
```

```
C*************************************************************
      integer idx(n,2),nd(n),it(ntud,2,nstr),ndf(nstr),id(n)
      integer iact(np),nostr(1),isub(1),gn(ncx,3)
      inct = n
      iprb=0
      lik=0.d0
      do 210 i = 1, n
        do 205 j = 1, np
           z(i,j) = 0.d0
  205 continue
  210 continue
      do 1 j=1,np
      s(j)=0.d0
      do 2 j1=1,np
      v(j,j1)=0.d0
    2 continue
    1 continue
c   loop over times (lci) within strata (lc2)
      nost=0
      do 996 lc2=1,nstr
      if (nt.le.n) then
       ntime=ntud
       else
       ntime=ndf(lc2)-nost
       endif
      do 98 lci=ntime,1,-1
c get which covariates are structurally 0 on time interval i
c nact is the # nonzero, iact(i) is the # of the ith nonzero covar
c ie iact(1)=2 means that the 2nd covariate is not structurally 0.
      if (nt.le.n) then
       l2=it(lci,2,lc2)
       l1=it(lci,1,lc2)
       else
       l2=lci+nost
       l1=lci+nost
       endif
      if (l1.le.0) go to 98
      call act(lci,l1,l2,idx,n,sv,ce,xx,nrx,ncx,iact,nact,np,
     &lc2,isub,dsub)
      xb1=0.d0
      xb1o=1
      ktmp=0
      do 11 j=1,np
        xb(j)=0.d0
        do 12 j1=j,np
          vt(j,j1)=0.d0
   12   continue
   11 continue
      l3=idx(ndf(lc2),2)
    4 do 10 k=l2,l1,-1
c k is ranging over unique failure times (largest to smallest)
c l to l4 are the failures at the kth unique failure time
c l to l3 are all obs at the kth unique failure but smaller than
c the next largest failure.
      l=idx(k,1)
      l4=l+nd(k)-1
CCC   ******************
      ncens = l3 - l4
      nf = l4 - l + 1
CCC   ******************
      l5=l4+1
      do 26 i=1,l4
c get covariates for obs i on time interval lci (gp)
```

```
          call eval(g,gp,b,lci,i,l,sv,ce,xx,nrx,ncx,n,np,lc2,nact,iact,
     &      isub,dsub)
          call eval8n(gn,lci,i,l,sv,ce,xx,nrx,ncx,n,np,lc2,nact,iact,
     &      isub,dsub,nfx)
c g=x'b
c          g=0
c          do 842 j=1,nact
c 842      g=g+gp(iact(j))*b(iact(j))
          lik=lik+g
          if (abs(g).gt.600) then
             iprb=1
             return
          endif
          u=dexp(g)
          xb1=xb1+u
          do 88 j=1,nact
   88     xb(iact(j))=xb(iact(j))+gp(iact(j))*u
          u=ktmp*u*(xb1/xb1o)
          do 19 j=1,nact
            s(iact(j))=s(iact(j))+gp(iact(j))
            u1=u*(gp(iact(j))-xb(iact(j))/xb1)
            do 19 j1=j,nact
              u2=gp(iact(j1))-xb(iact(j1))/xb1
              vt(iact(j),iact(j1))=vt(iact(j),iact(j1))+u1*u2
   19     continue
          xb1o=xb1
          ktmp=1
          do j = 1, nact
             z(inct-ncens,iact(j))=gp(iact(j))
C             z1(inct-ncens,iact(j))=gp(iact(j))
          enddo
          inct=inct-1
   26   continue
        if (15.le.13) then
C ***************************************************************
        ncsi = inct - ncens
        ict = 1
C ***************************************************************
          do 6 i=15,13
            call eval(g,gp,b,lci,i,l,sv,ce,xx,nrx,ncx,n,np,lc2,nact,iact,
     &        isub,dsub)
CCC             call eval8n(gn,lci,i,l,sv,ce,xx,nrx,ncx,n,np,lc2,nact,iact,
CCC     &       isub,dsub)
c            g=0
c            do 841 j=1,nact
c 841        g=g+gp(iact(j))*b(iact(j))
            if (abs(g).gt.600) then
               iprb=1
               return
            endif
            u=dexp(g)
            xb1=xb1+u
            do 8 j=1,nact
    8       xb(iact(j))=xb(iact(j))+gp(iact(j))*u
            u=ktmp*u*(xb1/xb1o)
            do 9 j=1,nact
              u1=u*(gp(iact(j))-xb(iact(j))/xb1)
              do 9 j1=j,nact
                u2=gp(iact(j1))-xb(iact(j1))/xb1
                vt(iact(j),iact(j1))=vt(iact(j),iact(j1))+u1*u2
    9       continue
            xb1o=xb1
            ktmp=1
```

```
        do j = 1, nact
C ************************************************************
           z(ncsi+ict+nf,iact(j))=gp(iact(j))
C           z1(ncsi+ict+nf,iact(j))=gp(iact(j))
C ************************************************************
        enddo
        ict = ict + 1
   6    continue
C ************************************************************
      inct = inct - ncens
C ************************************************************
      endif

c update score, likelihood, and inf:
c      if (xb1.le.0) write (6,118) xb1
c118   format ('xb1=',e12.4)
c110   format (6e12.4)
      lik=lik-nd(k)*dlog(xb1)
      do 18 j=1,nact
      s(iact(j))=s(iact(j))-nd(k)*xb(iact(j))/xb1
      do 20 j1=j,nact
  20  v(iact(j),iact(j1))=v(iact(j),iact(j1))+nd(k)*
     &vt(iact(j),iact(j1))/xb1
  18  continue
      l3=l-1
  10  continue
  98  continue
 101  format(i4,e20.10)
      nost=nostr(lc2)
 996  continue
c fill out symmetric information matrix:
      if (np.eq.1) go to 99
      do 56 j1=2,np
      do 56 j2=1,j1
  56  v(j1,j2)=v(j2,j1)
  99  return
      end

      subroutine strat(istr,s,c,d,nostr,nstr,it1,t1,no,nv,nnob,
     &iuti,tduti)
c subroutine to sort data on strata variable, and determine obs index
c of strata limits, also sorts on survival times within strata
c on input s and c are the survival and censoring vars,
c d is a double precision array containing the covariate data
c and istr gives the column of d containing the strat variable
c nnob is the row dimension of d in the calling program.  Actualy
c data matrix is no #obs x nv #vars.
c on output nstr is the # strata nostr(i) is the row # of the largest
c observation in strata i  (nostr must be at least nstr in lenght)
c t1 is double precision and it1 integer working vectors of length no
      double precision d(nnob,nv),t1(no),s(no),c(no)
      integer it1(no),nostr(1),iuti(no,2),tduti(no)
      if (istr.le.0) then
       nstr=1
       nostr(1)=no
      else
       do 214 i=1,no
       it1(i)=i
 214     t1(i)=d(i,istr)
       call sortg(t1,it1,1,no)
       call dperm(s,it1,1,no,t1)
       call dperm(c,it1,1,no,t1)
         do 217 j=1,nv
```

```
      call dperm(d(1,j),it1,1,no,t1)
217     continue
      nstr=0
      do 220 i=2,no
      if (d(i,istr).gt.d(i-1,istr)) then
         nstr=nstr+1
         nostr(nstr)=i-1
      endif
220     continue
      nstr=nstr+1
      nostr(nstr)=no
     endif
c  sort on survival times within strata
      l1=1
      do 225 l1=1,nstr
      l2=nostr(l1)
      do 15 i=l1,l2
      it1(i)=i
 15   continue
      call sortg(s,it1,l1,l2)
      call dperm(c,it1,l1,l2,t1)
      do 21 j=1,nv
      call dperm(d(1,j),it1,l1,l2,t1)
 21   continue
      call ut(l1,l2,s,c,iuti,nostr(nstr+l1),tduti,no)
      call uft(d,s,c,l1,iuti,nostr(nstr+l1),tduti,no,nnob,nv)
      l1=l2+1
225   continue
      return
      end

      subroutine ut(n1,n2,t,c,iuti,nuti,tduti,no)
c subroutine to determine unique times & # obs at each time
c on input t is a vector of times, and c is a vector of failure
c indicators. only observations #'ed n1-n2 (min-max) in t are considered
c no is the row dimension of iuti in the calling program.
c on output nuti is the # of unique values in t (between t(n1) and t(n2))
c iuti(i,1) and iuti(i,2) are the lower and upper index (in t) of the
c ith unique time, and tduti is the # failures (c=1) at the ith
c unique time.  t must be sorted in ascending order before calling ut.
c note that in this revised (8-15-90) version nuti is not the number
c of unique times, but rather the index such that the unique times
c are stored in indecies n1 to nuti
      double precision t(n2),c(n2),t1,t2
      integer iuti(no,2),tduti(1)
      l1=n1
      l2=n1-1
      t1=t(n1)
      iuti(n1,1)=n1
      t2=0
 10   t2=t2+c(l1)
      l1=l1+1
      if (l1.gt.n2) go to 11
      if (t(l1).eq.t1) go to 10
 11   l2=l2+1
      tduti(l2)=t2
      iuti(l2,2)=l1-1
      t2=0
      if (l1.le.n2) then
        t1=t(l1)
        go to 10
      endif
     nuti=l2
```

```
      do 20 i=n1+1,nuti
      iuti(i,1)=iuti(i-1,2)+1
   20 continue
      return
      end

      subroutine uft(d,s,c,n1,iuti,nuti,tduti,no,nnob,nv)
c subroutine to take output from ut and compress so only has lines
c for failures.  Also sorts data matrix d at each time so failures
c occur 1st
c nuti is the # unique times (output from ut) iuti gives the index range
c for the ith time interval and tduti the # failures at the ith unique
c time.  d is the data matrix, nnob is the row dimension of d
c in the calling program, and nv the # cols in d being used.
c kc is the col of d for failure indicator
c On output d has been sorted within unique times so failures occur
c first.  nuti is the # unique failure times, tduti is the # failures
c at each unique failure time, and iuti gives the index (row of d) for
c the obs >= the failure time and < the next failure time (the risk set
c increment) col 1 is the min and col 2 the max index.
      double precision d(nnob,nv),t1,c(1),s(1)
      integer iuti(no,2),tduti(nuti)
      do 10 k=n1,nuti
      if (tduti(k).gt.0) then
       nf=tduti(k)
       nn=iuti(k,2)-iuti(k,1)+1
       if (nn.gt.nf) then
        do 5 i=1,nf
        if (c(iuti(k,1)+i-1).le.0) then
    6    if (c(iuti(k,1)+nn-1).ge..99) then
          t1=c(iuti(k,1)+i-1)
          c(iuti(k,1)+i-1)=c(iuti(k,1)+nn-1)
          c(iuti(k,1)+nn-1)=t1
          do 7 j=1,nv
          t1=d(iuti(k,1)+i-1,j)
          d(iuti(k,1)+i-1,j)=d(iuti(k,1)+nn-1,j)
          d(iuti(k,1)+nn-1,j)=t1
    7     continue
          nn=nn-1
          if (nn.eq.nf) go to 8
         else
          nn=nn-1
          go to 6
         endif
        endif
    5   continue
    8   continue
       endif
      endif
   10 continue
      nutd=n1-1
      do 20 k=n1,nuti
      if (tduti(k).gt.0) then
       nutd=nutd+1
       iuti(nutd,1)=iuti(k,1)
       tduti(nutd)=tduti(k)
      endif
   20 continue
      do 25 k=n1+1,nutd
   25 iuti(k-1,2)=iuti(k,1)-1
      if (nutd.ge.n1) iuti(nutd,2)=iuti(nuti,2)
      nuti=nutd
      return
```

```
      end

      subroutine tint(ntud,t,s,c,itud,no,iuti,nuti,n1)
c subroutine to determine rows of iuti such that observations fall
c in time intervals given by t
c ntud is the # of time intervals, t(i) is the lower limit on interval
c i+1, (that is, the intervals are of the form I(j)=[t(j-1),t(j)).
c s is the vector of survival times and c the vector of failure
c indicators, no is the # of observations.
c  n1 is the lower limit index for the current strata
c iuti gives the indexes (row #'s) of the data matrix d such that
c iuti(j,1) is the row # of a failure time and iuti(j,2) is the
c row # of the largest observation smaller than the next largest
c failure time (ie the risk set increment at the jth unique failure
c time).  nuti is the # unique failure times, which is the # rows
c in iuti.  (nuti and iuti probably created by uft)
c on output itud gives the row #'s of iuti that correspond to failure
c times in each of the time intervals.  if there are no failures in
c a particular interval that row of itud will be -1 -1
      double precision s(no),c(no),t(ntud-1)
      integer itud(ntud,2),iuti(no,2)
      if (ntud.eq.1) then
       itud(1,1)=n1
       itud(1,2)=nuti
      else
      do 5 i=1,ntud
    5 itud(i,2)=0
      l1=1
      do 6 i=n1,nuti
    7 if (t(l1).gt.s(iuti(i,1))) then
      itud(l1,2)=itud(l1,2)+1
      else
      l1=l1+1
      if (l1.ge.ntud) go to 8
      go to 7
      endif
    6 continue
    8 l1=n1-1
      do 10 i=1,ntud-1
      if (itud(i,2).le.0) then
        itud(i,1)=-1
        itud(i,2)=-1
      else
        itud(i,1)=l1+1
        l1=l1+itud(i,2)
        itud(i,2)=l1
      endif
   10 continue
      if (l1.lt.nuti) then
      itud(ntud,1)=l1+1
      itud(ntud,2)=nuti
      else
      itud(ntud,1)=-1
      itud(ntud,2)=-1
      endif
      endif
      return
      end

      subroutine tint4(ntud,s,c,t,no,nfp)
c  subroutine to determine time intervals . The idea is to create ntud
c intervals with roughly equal #'s of failures. s (input) is the
c survival times and c (input) the failure indicator.
```

```
c  Data must be sorted on s ntud is the # of intervals (input).
c On output nfp(i) is the # of failures in
c the ith interval.  Ij is [t(j-1),t(j)).  (t(0)=0 is not given)
c t(j) is chosen so there will be a failure at t(j)
c That is, t(j) is actually the smallest failure in the j+1 interval.
c On input no is the # obs (length of s and c).
      double precision s(no),c(no),t(ntud-1),t1
      integer nfp(ntud)
      nft=0
      do 10 i=1,no
  10  nft=nft+c(i)
      l2=1
      t1=s(1)
      ntud1=ntud-1
      nfp(1)=0
      do 15 k=1,ntud1
c nfk is the target # failures for the kth interval
c continue advancing until total # >= nfk and find the
c next failure.  that next failure is t(k)
      nfk=max(int(nft/float(ntud-k+1)+.5),1)
  16  if (c(l2).eq.0) then
      l2=l2+1
      if (l2.gt.no) then
        ntud=k
c       write (6,100) ntud
c 100      format ('ntud reset in tint4 to', i5)
c          call intpr("ntud changed",12,ntud,1)
        go to 99
      else
        go to 16
      endif
      else
      if (s(l2).le.t1) then
        nfp(k)=nfp(k)+1
        l2=l2+1
        if (l2.gt.no) then
          ntud=k
c         write (6,100) ntud
c            call intpr("ntud changed",12,ntud,1)
          go to 99
        else
          go to 16
        endif
      else
        t1=s(l2)
        if (nfp(k).lt.nfk) then
          nfp(k)=nfp(k)+1
          l2=l2+1
          go to 16
        else
          t(k)=s(l2)
          nfp(k+1)=1
          l2=l2+1
            nft=nft-nfp(k)
        endif
      endif
      endif
  15  continue
      nfp(ntud)=nft
c      call dblepr("knots",5,t,ntud-1)
c      call intpr(" ",1,nfp,ntud)
  99  return
      end
```

```
      subroutine cholg(n,v,ch,nnvar,l2)
c subroutine to calc cholesky decomposition of a pd matrix (v)
c  on output ch will be upper triangular cholesky decomp
c  v=t(ch)*ch
c  nnvar and l2 are dimension of v and ch as specified in calling prog
c returns ch(1,1)=-1 if not pd
c ch and v can be the same matrix
c calls dpocog (linpack dpoco) which calls dpofag dscalg daxpyg ddotg dasumg
      double precision v(nnvar,n),ch(l2,n),rcond,z(200)
      if (n.gt.200) go to 520
      do 10 i=1,n
      do 10 j=i,n
   10 ch(i,j)=v(i,j)
      call dpocog(ch,l2,n,rcond,z,info)
c      call dblepr("rcond",5,rcond,1)
      if (rcond.lt.1.d-15.or.info.gt.0) go to 520
      do 40 i=2,n
      do 40 j=1,i-1
   40 ch(i,j)=0
      return
  520 continue
      ch(1,1)=-1
      return
      end

c*** from netlib, Sat Oct 12 09:38:00 EDT 1991 ***
c from linpack
c these are standard linpack routines.  Names changed so
c don't overwrite S routines of same names, since don't know if
c they are identical.
      subroutine dpocog(a,lda,n,rcond,z,info)
      integer lda,n,info
      double precision a(lda,1),z(1)
      double precision rcond
c
c     dpoco factors a double precision symmetric positive definite
c     matrix and estimates the condition of the matrix.
c
c     if  rcond  is not needed, dpofa is slightly faster.
c     to solve  a*x = b , follow dpoco by dposl.
c     to compute  inverse(a)*c , follow dpoco by dposl.
c     to compute  determinant(a) , follow dpoco by dpodi.
c     to compute  inverse(a) , follow dpoco by dpodi.
c
c     on entry
c
c        a       double precision(lda, n)
c                the symmetric matrix to be factored.  only the
c                diagonal and upper triangle are used.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix  r  so that  a = trans(r)*r
c                where  trans(r)  is the transpose.
c                the strict lower triangle is unaltered.
c                if  info .ne. 0 , the factorization is not complete.
```

```
c
c        rcond   double precision
c                an estimate of the reciprocal condition of  a .
c                for the system  a*x = b , relative perturbations
c                in  a  and  b  of size  epsilon  may cause
c                relative perturbations in  x  of size  epsilon/rcond .
c                if  rcond  is so small that the logical expression
c                           1.0 + rcond .eq. 1.0
c                is true, then  a  may be singular to working
c                precision.  in particular,  rcond  is zero  if
c                exact singularity is detected or the estimate
c                underflows.  if info .ne. 0 , rcond is unchanged.
c
c        z       double precision(n)
c                a work vector whose contents are usually unimportant.
c                if  a  is close to a singular matrix, then  z  is
c                an approximate null vector in the sense that
c                norm(a*z) = rcond*norm(a)*norm(z) .
c                if  info .ne. 0 , z  is unchanged.
c
c        info    integer
c                = 0  for normal return.
c                = k  signals an error condition.  the leading minor
c                     of order  k  is not positive definite.
c
c     linpack.  this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     linpack dpofa
c     blas daxpy,ddot,dscal,dasum
c     fortran dabs,dmax1,dreal,dsign
c
c     internal variables
c
      double precision ddotg,ek,t,wk,wkm
      double precision anorm,s,dasumg,sm,ynorm
      integer i,j,jm1,k,kb,kp1
c
c
c     find norm of a using only upper half
c
      do 30 j = 1, n
         z(j) = dasumg(j,a(1,j),1)
         jm1 = j - 1
         if (jm1 .lt. 1) go to 20
         do 10 i = 1, jm1
            z(i) = z(i) + dabs(a(i,j))
   10    continue
   20    continue
   30 continue
      anorm = 0.0d0
      do 40 j = 1, n
         anorm = dmax1(anorm,z(j))
   40 continue
c
c     factor
c
      call dpofag(a,lda,n,info)
      if (info .ne. 0) go to 180
c
c        rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
```

```
c        estimate = norm(z)/norm(y) where  a*z = y  and  a*y = e .
c        the components of  e  are chosen to cause maximum local
c        growth in the elements of w  where  trans(r)*w = e .
c        the vectors are frequently rescaled to avoid overflow.
c
c        solve trans(r)*w = e
c
         ek = 1.0d0
         do 50 j = 1, n
            z(j) = 0.0d0
   50    continue
         do 110 k = 1, n
            if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
            if (dabs(ek-z(k)) .le. a(k,k)) go to 60
               s = a(k,k)/dabs(ek-z(k))
               call dscalg(n,s,z,1)
               ek = s*ek
   60       continue
            wk = ek - z(k)
            wkm = -ek - z(k)
            s = dabs(wk)
            sm = dabs(wkm)
            wk = wk/a(k,k)
            wkm = wkm/a(k,k)
            kp1 = k + 1
            if (kp1 .gt. n) go to 100
               do 70 j = kp1, n
                  sm = sm + dabs(z(j)+wkm*a(k,j))
                  z(j) = z(j) + wk*a(k,j)
                  s = s + dabs(z(j))
   70          continue
               if (s .ge. sm) go to 90
                  t = wkm - wk
                  wk = wkm
                  do 80 j = kp1, n
                     z(j) = z(j) + t*a(k,j)
   80             continue
   90          continue
  100       continue
            z(k) = wk
  110    continue
         s = 1.0d0/dasumg(n,z,1)
         call dscalg(n,s,z,1)
c
c        solve r*y = w
c
         do 130 kb = 1, n
            k = n + 1 - kb
            if (dabs(z(k)) .le. a(k,k)) go to 120
               s = a(k,k)/dabs(z(k))
               call dscalg(n,s,z,1)
  120       continue
            z(k) = z(k)/a(k,k)
            t = -z(k)
            call daxpyg(k-1,t,a(1,k),1,z(1),1)
  130    continue
         s = 1.0d0/dasumg(n,z,1)
         call dscalg(n,s,z,1)
c
         ynorm = 1.0d0
c
c        solve trans(r)*v = y
c
```

```
      do 150 k = 1, n
         z(k) = z(k) - ddotg(k-1,a(1,k),1,z(1),1)
         if (dabs(z(k)) .le. a(k,k)) go to 140
            s = a(k,k)/dabs(z(k))
            call dscalg(n,s,z,1)
            ynorm = s*ynorm
  140    continue
         z(k) = z(k)/a(k,k)
  150 continue
      s = 1.0d0/dasumg(n,z,1)
      call dscalg(n,s,z,1)
      ynorm = s*ynorm
c
c     solve r*z = v
c
      do 170 kb = 1, n
         k = n + 1 - kb
         if (dabs(z(k)) .le. a(k,k)) go to 160
            s = a(k,k)/dabs(z(k))
            call dscalg(n,s,z,1)
            ynorm = s*ynorm
  160    continue
         z(k) = z(k)/a(k,k)
         t = -z(k)
         call daxpyg(k-1,t,a(1,k),1,z(1),1)
  170 continue
c     make znorm = 1.0
      s = 1.0d0/dasumg(n,z,1)
      call dscalg(n,s,z,1)
      ynorm = s*ynorm
c
      if (anorm .ne. 0.0d0) rcond = ynorm/anorm
      if (anorm .eq. 0.0d0) rcond = 0.0d0
  180 continue
      return
      end

      subroutine dpofag(a,lda,n,info)
      integer lda,n,info
      double precision a(lda,1)
c
c     dpofa factors a double precision symmetric positive definite
c     matrix.
c
c     dpofa is usually called by dpoco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for dpoco) = (1 + 18/n)*(time for dpofa) .
c
c     on entry
c
c        a       double precision(lda, n)
c                the symmetric matrix to be factored.  only the
c                diagonal and upper triangle are used.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix  r  so that  a = trans(r)*r
```

```
c                where  trans(r)  is the transpose.
c                the strict lower triangle is unaltered.
c                if  info .ne. 0 , the factorization is not complete.
c
c        info    integer
c                = 0  for normal return.
c                = k  signals an error condition.  the leading minor
c                     of order  k  is not positive definite.
c
c     linpack.  this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas ddot
c     fortran dsqrt
c
c     internal variables
c
      double precision ddotg,t
      double precision s
      integer j,jm1,k
c     begin block with ...exits to 40
c
c
         do 30 j = 1, n
            info = j
            s = 0.0d0
            jm1 = j - 1
            if (jm1 .lt. 1) go to 20
            do 10 k = 1, jm1
               t = a(k,j) - ddotg(k-1,a(1,k),1,a(1,j),1)
               t = t/a(k,k)
               a(k,j) = t
               s = s + t*t
   10       continue
   20       continue
            s = a(j,j) - s
c     ......exit
            if (s .le. 0.0d0) go to 40
            a(j,j) = dsqrt(s)
   30    continue
         info = 0
   40 continue
      return
      end

      subroutine  dscalg(n,da,dx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision da,dx(1)
      integer i,incx,m,mp1,n,nincx
c
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
```

```
        dx(i) = da*dx(i)
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end

      double precision function dasumg(n,dx,incx)
c
c     takes the sum of the absolute values.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dtemp
      integer i,incx,m,mp1,n,nincx
c
      dasumg = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dtemp = dtemp + dabs(dx(i))
   10 continue
      dasumg = dtemp
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,6)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dabs(dx(i))
   30 continue
      if( n .lt. 6 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,6
        dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
     *  + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
```

```
   50 continue
   60 dasumg = dtemp
      return
      end

      subroutine daxpyg(n,da,dx,incx,dy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dy(1),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end

      double precision function ddotg(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dy(1),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddotg = 0.0d0
```

```
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddotg = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     *   dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddotg = dtemp
      return
      end

      subroutine solve(n,ch,b,x,nnvar)
      double precision ch(nnvar,n),b(n),x(n)
c  ch is upper triang chol decomp, b (input) is the rhs, x (output)
c  is the solution
c
c  first solve t(ch)*x=b
      call slv1(n,ch,b,x,nnvar)
c  then solve ch*x=xold
      call slv2(n,ch,x,x,nnvar)
      return
      end

      subroutine sortg(v,a,ii,jj)
c
c     puts into a the permutation vector which sorts v into
c     increasing order.  only elements from ii to jj are considered.
c     arrays iu(k) and il(k) permit sorting up to 2**(k+1)-1 elements
c     v is returned sorted
c     this is a modification of cacm algorithm #347 by r. c. singleton,
c     which is a modified hoare quicksort.
c
c on input a has integers ii to jj in components ii to jj
c
      dimension a(jj),v(1),iu(20),il(20)
```

```
      integer t,tt
      integer a
      double precision v,vt,vtt
      m=1
      i=ii
      j=jj
10    if (i.ge.j) go to 80
20    k=i
      ij=(j+i)/2
      t=a(ij)
      vt=v(ij)
      if (v(i).le.vt) go to 30
      a(ij)=a(i)
      a(i)=t
      t=a(ij)
      v(ij)=v(i)
      v(i)=vt
      vt=v(ij)
30    l=j
      if (v(j).ge.vt) go to 50
      a(ij)=a(j)
      a(j)=t
      t=a(ij)
      v(ij)=v(j)
      v(j)=vt
      vt=v(ij)
      if (v(i).le.vt) go to 50
      a(ij)=a(i)
      a(i)=t
      t=a(ij)
      v(ij)=v(i)
      v(i)=vt
      vt=v(ij)
      go to 50
40    a(l)=a(k)
      a(k)=tt
      v(l)=v(k)
      v(k)=vtt
50    l=l-1
      if (v(l).gt.vt) go to 50
      tt=a(l)
      vtt=v(l)
60    k=k+1
      if (v(k).lt.vt) go to 60
      if (k.le.l) go to 40
      if (l-i.le.j-k) go to 70
      il(m)=i
      iu(m)=l
      i=k
      m=m+1
      go to 90
70    il(m)=k
      iu(m)=j
      j=l
      m=m+1
      go to 90
80    m=m-1
      if (m.eq.0) return
      i=il(m)
      j=iu(m)
90    if (j-i.gt.10) go to 20
      if (i.eq.ii) go to 10
      i=i-1
```

```
 100  i=i+1
      if (i.eq.j) go to 80
      t=a(i+1)
      vt=v(i+1)
      if (v(i).le.vt) go to 100
      k=i
 110  a(k+1)=a(k)
      v(k+1)=v(k)
      k=k-1
      if (vt.lt.v(k)) go to 110
      a(k+1)=t
      v(k+1)=vt
      go to 100
      end

      subroutine dperm(a,ia,nl,nu,work)
      double precision a(nu),work(nu)
      integer ia(nu)
      do 10 i=nl,nu
  10  work(i)=a(i)
      do 11 i=nl,nu
  11  a(i)=work(ia(i))
      return
      end

      subroutine adisw(n1,n2,n,v,v2,lb)
c  calcs adjusted information.  v is the full information matrix
c  (v is nxn), n1 and n2 are min and max index of the block of interest
c  lb is actual row dimension of v and v2.  final result will be in
c  elements 1:(n2-n1+1)*1:(n2-n1+1) of v2.  v2 must also be nxn.
      double precision v(lb,n),v2(lb,n)
      np=n2-n1+1
      if (np.ge.n) then
      do 10 i=1,n
      do 10 j=1,n
  10    v2(i,j)=v(i,j)
      return
      endif
      if (n1.le.1) then
      call gsweep(n1,n2,n2+1,n,v,v2,lb)
      do 12 k=n2+2,n
  12    call gsweep(n1,n2,k,n,v2,v2,lb)
      else
        call gsweep(n1,n2,1,n,v,v2,lb)
        do 14 k=2,n1-1
  14    call gsweep(n1,n2,k,n,v2,v2,lb)
        do 16 k=n2+1,n
  16    call gsweep(n1,n2,k,n,v2,v2,lb)
      endif
      do 20 i=1,np
      i2=i+n1-1
      v2(i,i)=v2(i2,i2)
      do 21 j=i+1,np
      j2=j+n1-1
      v2(j,i)=v2(i2,j2)
  21  v2(i,j)=v2(i2,j2)
  20  continue
      return
      end

      subroutine gsweep(n1,n2,l,n,v,v2,lb)
c  note that this is intended for use with adjusted inf routine:
c  it is assumed that l is not in [n1,n2]
```

```
c  v is the matrix to be swept on the lth component, result put in v2.
c  v and v2 can be the same.  assumes symmetric matrices, only upper
c  triangular part is used.  assumes that sweeping is proceding in
c  order, so that only elements from l to n (+[n1-n2]) need to be swept.
c
      double precision v(lb,n),v2(lb,n)
      v2(l,l)=-1/v(l,l)
      do 10 i=l+1,n
   10 v2(l,i)=v(l,i)*v2(l,l)
      do 11 i=1,l-1
   11 v2(i,l)=v(i,l)*v2(l,l)
      do 12 i=l+1,n
      do 13 j=1,l-1
   13 v2(j,i)=v(j,i)+v2(j,l)*v2(l,i)/v2(l,l)
      do 14 j=i,n
   14 v2(i,j)=v(i,j)+v2(l,i)*v2(l,j)/v2(l,l)
   12 continue
      if (l.ge.n1) then
      do 22 i=n1,n2
      do 22 j=i,n2
   22   v2(i,j)=v(i,j)+v2(i,l)*v2(j,l)/v2(l,l)
      endif
      return
      end

      SUBROUTINE eigen(A,N,NP,D,E,iopt,iflg)
      implicit double precision (a - h, o - z)
c  subroutine to calculate eigenvalues and eigenvectors
c  based on Numerical Recipies routines
c  on input a is a symmetric matrix, nxn, but dimensioned npxn
c in the calling program.
c a is destroyed, in this routine, but the call to tql2 writes
c the eigenvectors to a and the eigen values to d.  The original
c matrix can then be calculated as ADA', where D is the diag matrix
c with the eigenvalues on the diagonal.  Note that the eigenvalues
c are not sorted.
c iflg=1 means the iteration in tql2 failed to converge.
c this routine reduces a to tridiag form. tql2 actually calculates
c eigenvalues and eigenvectors.
c iopt<=0 only eigenvalues calculated
c iopt>0 eigenvalues & vectors calculated
      DIMENSION A(NP,N),D(1),E(1)
      IF(N.GT.1)THEN
        DO 18 I=N,2,-1
          L=I-1
          H=0.d0
          SCALE=0.d0
          IF(L.GT.1)THEN
            DO 11 K=1,L
              SCALE=SCALE+ABS(A(I,K))
11          CONTINUE
            IF(SCALE.EQ.0.d0)THEN
              E(I)=A(I,L)
            ELSE
              DO 12 K=1,L
                A(I,K)=A(I,K)/SCALE
                H=H+A(I,K)**2
12            CONTINUE
              F=A(I,L)
              G=-SIGN(SQRT(H),F)
              E(I)=SCALE*G
              H=H-F*G
              A(I,L)=F-G
```

```
            F=0.d0
            DO 15 J=1,L
              A(J,I)=A(I,J)/H
              G=0.d0
              DO 13 K=1,J
                G=G+A(J,K)*A(I,K)
13            CONTINUE
              IF(L.GT.J)THEN
                DO 14 K=J+1,L
                  G=G+A(K,J)*A(I,K)
14              CONTINUE
              ENDIF
              E(J)=G/H
              F=F+E(J)*A(I,J)
15          CONTINUE
            HH=F/(H+H)
            DO 17 J=1,L
              F=A(I,J)
              G=E(J)-HH*F
              E(J)=G
              DO 16 K=1,J
                A(J,K)=A(J,K)-F*E(K)-G*A(I,K)
16            CONTINUE
17          CONTINUE
          ENDIF
        ELSE
          E(I)=A(I,L)
        ENDIF
        D(I)=H
18    CONTINUE
      ENDIF
      if (iopt.gt.0) then
        D(1)=0.d0
        E(1)=0.d0
        DO 23 I=1,N
          L=I-1
          IF(D(I).NE.0.d0)THEN
            DO 21 J=1,L
              G=0.d0
              DO 19 K=1,L
                G=G+A(I,K)*A(K,J)
19            CONTINUE
              DO 20 K=1,L
                A(K,J)=A(K,J)-G*A(K,I)
20            CONTINUE
21          CONTINUE
          ENDIF
          D(I)=A(I,I)
          A(I,I)=1.d0
          IF(L.GE.1)THEN
            DO 22 J=1,L
              A(I,J)=0.d0
              A(J,I)=0.d0
22          CONTINUE
          ENDIF
23      CONTINUE
      else
      E(1)=0.d0
      DO 25 I=1,N
        D(I)=A(I,I)
25      CONTINUE
      endif
      call TQL2g(D,E,N,np,a,iopt,iflg)
```

```
      RETURN
      END

      SUBROUTINE TQL2g(D,E,N,np,z,iopt,iflg)
c subroutine for finding eigenvalues of a tridiagonal matrix
c symmetric matrix reduced to tridiagonal form with tred2 (d and e
c are output from tred2).  n is the length of d and e and portion of
c z that is used.  np is actual dim of z in calling program.
c On output d contains the eigenvalues. iflg=1 if
c iteration failed to converge.
c if iopt>0 then on output z contains the eigenvectors
      implicit double precision (a - h, o - z)
      DIMENSION D(1),E(1),Z(NP,N)
      iflg=0
      IF (N.GT.1) THEN
        DO 11 I=2,N
          E(I-1)=E(I)
11      CONTINUE
        E(N)=0.d0
        DO 15 L=1,N
          ITER=0
1         DO 12 M=L,N-1
            DD=ABS(D(M))+ABS(D(M+1))
            IF (ABS(E(M))+DD.EQ.DD) GO TO 2
12        CONTINUE
          M=N
2         IF(M.NE.L)THEN
            IF(ITER.GE.30) then
            iflg=1
            go to 15
          endif
            ITER=ITER+1
            G=(D(L+1)-D(L))/(2.d0*E(L))
            R=SQRT(G**2+1.d0)
            G=D(M)-D(L)+E(L)/(G+SIGN(R,G))
            S=1.d0
            C=1.d0
            P=0.d0
            DO 14 I=M-1,L,-1
              F=S*E(I)
              B=C*E(I)
              IF(ABS(F).GE.ABS(G))THEN
                C=G/F
                R=SQRT(C**2+1.d0)
                E(I+1)=F*R
                S=1.d0/R
                C=C*S
              ELSE
                S=F/G
                R=SQRT(S**2+1.d0)
                E(I+1)=G*R
                C=1.d0/R
                S=S*C
              ENDIF
              G=D(I+1)-P
              R=(D(I)-G)*S+2.d0*C*B
              P=S*R
              D(I+1)=G+P
              G=C*R-B
            if (iopt.gt.0) then
                DO 13 K=1,N
                  F=Z(K,I+1)
                  Z(K,I+1)=S*Z(K,I)+C*F
```

```
                Z(K,I)=C*Z(K,I)-S*F
13             CONTINUE
              endif
14          CONTINUE
            D(L)=D(L)-P
            E(L)=G
            E(M)=0.d0
            GO TO 1
          ENDIF
15      CONTINUE
      ENDIF
      RETURN
      END

      subroutine pdi(n,v,ch,nnvar,l2)
c subroutine to invert a pd matrix. v=input matrix
c  on output ch will be inverse
c  returns ch(1,1)=-1 if not pd
      double precision v(nnvar,n),ch(l2,n),tmp
      if (v(1,1).le.0) go to 520
      if (n.eq.1) then
       ch(1,1)=1/v(1,1)
       return
      endif
      call cholg(n,v,ch,nnvar,l2)
      if (ch(1,1).le.0) go to 520
      n1=n-1
      ch(n,n)=1/ch(n,n)
      do 40 i=n1,1,-1
      ch(i,i)=1/ch(i,i)
      i1=i+1
      do 41 j=i1,n
  41  ch(i,j)=ch(i,j)*ch(i,i)
  40  continue
       do 45 j=n,2,-1
       j2=j-1
       do 46 k=j2,1,-1
       do 47 j1=j,n
  47     ch(j1,k)=ch(j1,k)-ch(k,j)*ch(j1,j)
  46     continue
  45     continue
      do 50 i=1,n
      do 52 j=i,n
      tmp=0
      do 53 k=j,n
  53  tmp=tmp+ch(k,i)*ch(k,j)
      ch(i,j)=tmp
  52  continue
  50  continue
       do 55 i=2,n
       i1=i-1
       do 56 j=1,i1
  56     ch(i,j)=ch(j,i)
  55     continue
      return
 520  continue
c 520  write (6,101)
 101  format('matrix not pd in pdi')
      ch(1,1)=-1
      return
      end

      subroutine pdi2(n,v,ch,nnvar,nv2)
```

```
c subroutine to factor a pd matrix, and invert the factorization
c  . v=input matrix
c  on output ch will be inverse square root of v
c  returns ch(1,1)=-1 if not pd
      double precision v(nnvar,n),ch(nv2,n)
      if (v(1,1).le.0) go to 520
      if (n.eq.1) then
       ch(1,1)=1/v(1,1)
       return
      endif
      call cholg(n,v,ch,nnvar,nv2)
      if (ch(1,1).le.0) go to 520
      n1=n-1
      ch(n,n)=1/ch(n,n)
      do 40 i=n1,1,-1
      ch(i,i)=1/ch(i,i)
      i1=i+1
      do 41 j=i1,n
 41   ch(i,j)=ch(i,j)*ch(i,i)
 40   continue
       do 45 j=n,2,-1
       j2=j-1
       do 46 k=j2,1,-1
       do 47 j1=j,n
 47      ch(j1,k)=ch(j1,k)-ch(k,j)*ch(j1,j)
 46      continue
 45      continue
       do 55 i=2,n
       i1=i-1
       do 56 j=1,i1
         ch(j,i)=ch(i,j)
 56      ch(i,j)=0
 55      continue
      return
 520  continue
c write (6,101)
 101  format('matrix not pd')
      ch(1,1)=-1
      return
      end

      subroutine slv1(n,ch,b,x,nnvar)
      double precision ch(nnvar,n),b(n),x(n)
c  ch is upper triang chol decomp, b (input) is the rhs, x (output)
c  is the solution
c  this routine solves t(ch)*x=b
      x(1)=b(1)/ch(1,1)
      if (n.gt.1) then
      do 10 i=2,n
      i1=i-1
      x(i)=b(i)
      do 11 j=1,i1
 11   x(i)=x(i)-ch(j,i)*x(j)
      x(i)=x(i)/ch(i,i)
 10   continue
      endif
      return
      end

      subroutine slv2(n,ch,b,x,nnvar)
      double precision ch(nnvar,n),b(n),x(n)
c  ch is upper triang chol decomp, b (input) is the rhs, x (output)
c  is the solution
```

```
c  this routine solves ch*x=b
      x(n)=b(n)/ch(n,n)
      if (n.gt.1) then
      n1=n-1
      do 12 i=n1,1,-1
      l1=i+1
      x(i)=b(i)
      do 13 j=l1,n
   13 x(i)=x(i)-ch(i,j)*x(j)
      x(i)=x(i)/ch(i,i)
   12 continue
      endif
      return
      end
      subroutine extrct(v,np,lv,l1,l2,a,la)
      double precision v(lv,np),a(la,1)
c v input: np*np matrix
c l1 & l2: rows and columns from 1 to l1 and l2 to np are
c extracted from v and stored in a (a and v can be the same
c matrix.
      do 10 i=1,l1
      do 10 j=1,l1
   10 a(i,j)=v(i,j)
      k=l1
      do 20 j=l2,np
      k=k+1
      do 20 i=1,l1
      a(i,k)=v(i,j)
      a(k,i)=v(j,i)
   20 continue
      k=l2-l1-1
      do 30 i=l2,np
      do 30 j=l2,np
   30 a(i-k,j-k)=v(i,j)
      return
      end

      subroutine matop5(vv,np,vc,n2,l1,l2,v5,n4,v6,v7,l6)
      double precision vv(np,np),vc(n2,n2),v5(l6,n4),v6(l6,n4),
     &v7(n4,n4)
c l6 is the actual row dim of v5 and v6 in the calling program
c l1 to l2 are the spline terms, vv the unpenalized inf
c n4=l2-l1+1,n2=np-n4, vc is the ut chol decomp of v^star_beta,beta,
c where v^star is the pen inf, but without the pen applied to l1 to l2,
c and 'beta' means the parameters not in l1 to l2.  v5,v6, and v7 are
c as dimensioned.  On output v5 contains the appropriate terms for
c var_thetatheta|beta, & v6 the appropriate terms for v^star_thetatheta
c |beta
      do 10 i=1,l1-1
      do 10 j=l1,l2
   10 v5(i,j-l1+1)=vv(i,j)
      k=l2-l1+1
      do 20 i=l2+1,np
      do 20 j=l1,l2
   20 v5(i-k,j-l1+1)=vv(i,j)
      do 30 i=1,n4
      call solve(n2,vc,v5(1,i),v6(1,i),n2)
   30 continue
c calc v7=v^star_theta,theta|beta
      do 11 i=1,n4
      do 11 j=i,n4
      v7(i,j)=vv(i+l1-1,j+l1-1)
      do 13 k=1,n2
```

```
  13  v7(i,j)=v7(i,j)-v5(k,i)*v6(k,j)
      v7(j,i)=v7(i,j)
  11  continue
c set vc=v_beta,beta
      call extrct(vv,np,np,l1-1,l2+1,vc,n2)
c and set v5=(approp terms)-t(v6)*vc*v6
      do 22 i=1,n4
      do 22 j=i,n4
      v5(i,j)=2*v7(i,j)-vv(i+l1-1,j+l1-1)
      do 23 ii=1,n2
      do 23 jj=1,n2
  23  v5(i,j)=v5(i,j)+v6(ii,i)*v6(jj,j)*vc(ii,jj)
      v5(j,i)=v5(i,j)
  22  continue
      do 25 i=1,n4
      do 25 j=1,n4
  25  v6(i,j)=v7(i,j)
      return
      end

      subroutine pencb(nk,wk,penm,mr)
c nk is # knots, wk is augmented knot seq.
c This subroutine calculates the penalty matrix for integrated squared
c second derivative penalty. The first col of penm will have the main
c diagonal, the 2nd col the next diag ... the 4th col the 3rd diag from
c the main.
c mr is the actual row dim of penm
      double precision wk(-2:nk+3),penm(mr,4),t1(4),tu(4),a(4),b(4)
      double precision t1,t2,t3,t4
      do 10 i=1,nk+4
      do 10 j=1,4
  10  penm(i,j)=0
      t1(1)=6/((wk(1)-wk(-2))*(wk(1)-wk(-1)))
      t1(2)=-6/(wk(1)-wk(-1))*(1/(wk(1)-wk(-2))+1/(wk(2)-wk(-1)))
      t1(3)=6/((wk(2)-wk(-1))*(wk(1)-wk(-1)))
      t1(4)=0
      do 20 l=1,nk+1
c t1 has the value of the 2nd deriv at the lower endpoint of the lth
c interval, tu at the upper (2nd derivs of basis fcns are linear on
c each interval
      t1=wk(l)-wk(l-1)
      t4=wk(l)+wk(l-1)
      t2=t1*t4/2
      t3=t1*(t4*t4-wk(l)*wk(l-1))/3
c note that t3 is (wk(l)^3-wk(l-1)^3)/3
      tu(1)=0
      tu(2)=6/((wk(l+1)-wk(l-2))*(wk(l+1)-wk(l-1)))
      tu(3)=-6/(wk(l+1)-wk(l-1))*(1/(wk(l+1)-wk(l-2))+
     &1/(wk(l+2)-wk(l-1)))
      tu(4)=6/((wk(l+2)-wk(l-1))*(wk(l+1)-wk(l-1)))
c calc slopes and intercepts on interval l:
      do 21 j=1,4
      b(j)=(tu(j)-t1(j))/(wk(l)-wk(l-1))
      a(j)=tu(j)-b(j)*wk(l)
  21  continue
      do 22 j=1,4
      l1=l+j-1
      j2=j-1
      do 22 k=j,4
      penm(l1,k-j2)=penm(l1,k-j2)+2*(a(j)*a(k)*t1+(a(j)*b(k)+
     &a(k)*b(j))*t2+b(j)*b(k)*t3)
  22  continue
      t1(1)=tu(2)
```

```
      tl(2)=tu(3)
      tl(3)=tu(4)
      tl(4)=0
   20 continue
      return
      end

      subroutine penh(nb,penm,mr,md,h,nmin,ih,alpha)
c penm is the penalty matrix,  nb the actual
c # basis functions used, h the neg sec deriv matrix
c nmin the entry in h where spline params begin.  alpha*P
c is added to the submatrix bounded by h(nm,nm)and h(nm+nb-1,
c nm+nb-1)
c mr is the actual row dimension of penm, md is the # nonzero
c diagonals in the penalty matrix
      double precision penm(mr,md),h(ih,ih),alpha,u
      do 10 i=1,nb
      i1=nmin+i-1
      h(i1,i1)=h(i1,i1)+alpha*penm(i,1)
      if (i.lt.nb) then
        m2=min(nb,i+md-1)
        l=1
        do 12 j=i+1,m2
        j1=nmin+j-1
        l=l+1
        u=alpha*penm(i,l)
        h(i1,j1)=h(i1,j1)+u
        h(j1,i1)=h(j1,i1)+u
   12   continue
      endif
   10 continue
      return
      end

      subroutine gdf(ntud3,vv,v3,var,vvi,lb,df,alpha,v4,mindf,md,mr,
     &iopt)
c change made 10-26-91 iopt returns 0 if ok, 1 if singular or other
c  problems
c change made 5-5-91, fast alg no longer an option (was found to be
c unstable,iopt not really used anymore, but left in for compatibility
c with calling programs)
c change made 3-21-91, must input a starting value for alpha:
c ntud3 is the # parameters,
c vv is the penalty matrix (mr*md), v3 is the unpenalized -2nd deriv,
c var is the actual var matrix (both have row dim lb).  var and v3 can be
c the same matrix (vv v3 and var are not changed)
c v4 is a working matrix with dim ntud3*ntud3.
c vvi a vector of length ntud3
c df is the target df (input) on output df=attained df,
c alpha is the smp (output)
c mindf is the minimum degree of freedom possible for this spline/pen
c combination.  (note that this is double precision)
c md is the number of nonzero diagonals in the penalty matrix
c mr the actual row dim of pen mat
c calls cholg, solve,(and degf2) and penh.
c
      double precision vv(mr,md),vvi(ntud3),v3(lb,lb),df
      double precision al,au,ap,del,var(lb,lb),alpha
      double precision fl,fu,fp,v4(ntud3,ntud3),mindf,eps
      eps=1.d-3
      mit=100
      nit=0
      if (df.ge.ntud3) then
```

```
        alpha=0
        df=ntud3
        return
      else if (df.le.mindf) then
        df=mindf+.05d0
      endif
c calculate smoothing parameter to get appropriate degrees of freedom
c  start nonpd algorithm:
 200  iopt=0
      ap=alpha
      fu=df+1
 202  call degf2(vv,v3,var,v4,vvi,ntud3,ap,lb,fp,md,mr)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      nit=nit+1
      if (abs(fp-df).lt.eps) go to 250
      if (nit.gt.mit) then
         iopt=1
         go to 250
      endif
      if (fp.gt.df) then
        al=ap
        fl=fp
        go to 205
      else
        au=ap
        fu=fp
        ap=ap/4
        go to 202
      endif
 205  if (fu.lt.df) go to 208
      au=al
      del=4
 206  au=au*del
      call degf2(vv,v3,var,v4,vvi,ntud3,au,lb,fu,md,mr)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      nit=nit+1
      if (abs(fu-df).lt.eps) then
        ap=au
        fp=fu
        go to 250
      endif
      if (nit.gt.mit) then
         iopt=1
         go to 250
      endif
      if (fu.gt.df) then
        al=au
        fl=fu
        go to 206
      endif
 208  ap=(al+au)/2
      call degf2(vv,v3,var,v4,vvi,ntud3,ap,lb,fp,md,mr)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      nit=nit+1
```

```
      if (abs(fp-df).lt.eps) go to 250
      if (nit.gt.mit) then
         iopt=1
         go to 250
      endif
      if (fp.gt.df) then
       al=ap
       fl=fp
      else
       au=ap
       fu=fp
      endif
      go to 208
 250  alpha=ap
      df=fp
 110  format(4e18.10)
c      write (70,*) nit
      return
      end

      subroutine degf2(vv,v3,var,v4,vvi,ntud3,a0,lb,t1,md,mr)
c  on output t1 is the trace (=df)
c  vv is pen v3 is inf v4 vvi are work var is the true var
      double precision vv(mr,md),v3(lb,lb),v4(ntud3,ntud3),a0
      double precision vvi(ntud3),t1,var(lb,lb)
      do 10 i=1,ntud3
      do 10 j=1,ntud3
      v4(i,j)=v3(i,j)
  10  continue
      call penh(ntud3,vv,mr,md,v4,1,ntud3,a0)
      call cholg(ntud3,v4,v4,ntud3,ntud3)
      if (v4(1,1).lt.0) then
      call dblepr("not pd in degf",14,v4(1,1),1)
 143   format ('matrix not pd in degf')
       return
       endif
      t1=0
      do 20 i=1,ntud3
      call solve(ntud3,v4,var(1,i),vvi,ntud3)
      t1=t1+vvi(i)
  20  continue
      return
      end

      subroutine penlik(nb,beta,penm,mr,md,lik,s,alpha)
c given a penalty matrix penm and parameter values beta,
c subtracts penalty terms from liklihood and score
c the beginning index of beta and s (the score) in the
c call should correspond to where the spline terms begin
c ie call penlik(...beta(7)...s(7)...) if the spline is
c stored consecutively beginning with the 7th component
c nk is the number of knots, nb is the number
c of basis fcns actually in use, alpha is the value of the
c smoothing parameter.
      double precision penm(mr,md),beta(1),s(1),lik,alpha,u
      do 10 i=1,nb
      u=0
      m1=max(1,i-md+1)
      m2=min(nb,i+md-1)
      l=0
      do 12 j=i,m2
      l=l+1
      u=u+penm(i,l)*beta(j)
```

```
  12  continue
      if (m1.lt.i) then
        l=i-m1+2
        do 11 j=m1,i-1
        l=l-1
        u=u+beta(j)*penm(j,l)
  11    continue
      endif
      s(i)=s(i)-alpha*u
      lik=lik-alpha*u*beta(i)/2
  10  continue
      return
      end

      subroutine sph(nov,s,c,x,tknot,beta,lik0,lik,sb,vv,
     &iwork,dsub,isub,testr,eig,eps,z,exz,s0,s1,ts1,w,
     &wp,ts1p,id,s1p,z1,z2,gn)
      double precision s(1),c(1),x(1),tknot(1),times(1),beta(1),lik0
      double precision lik(2),eps,sb(1),vv(1),dsub(1),testr(1),eig(1)
      double precision uplik
      double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C>>>>>
      double precision wp(1),ts1p(1),s1p(1),z2(1)
      integer id(1),gn(1)
C<<<<<
      integer nov(15),iwork(1),isub(1)
      common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph
      common /up/ uplik
c
c all covariates get included in the first
c nfx columns of x.  nsx then gives the number that will have spline
c components.  That is, splines will be linear+B-spline terms.  So
c for each spline variable need nknot+2 B-spline terms.
c
c dsub must have length at least 3*nsx+4*nsx*(nknot+4)+2*(nknot+3)^2
c +np*np+max(no+5*np,(nknot+3)*((nknot+3)))+(if ksmopt>0) (2*(nknot+3)*nkk
c iwork must have length at least 3*no+max(no,np)+1+2*nstr+2*nstr (since ntud=1)
c where nstr is the # strata.
c
c nov parameters:
c   1=no, 2=nrx, 3=ncx, 4=istr (col # of x for strata, -1 if none,
c   5=maxiter, 6=knot optiont (1 use knots provided, 0 program calculates),
c   7=smoothing option (<0 use input smoothing params, 0 calc smoothing
c   param only in first iteration, >0 recalc after each iteration)
c   8=nfx total # covariates, 9=nsx, the # using splines
c   10=analysis option (<=0 estimates only, 1 est & var, >1 est, var & test),
c   12=nknot, 13=nest
c
c isub(7) to isub(6+nsx) must have the col #'s in x of the spline terms
c testr must have length nsx*6, where nsx is nov(9)
c length(eig)=2*nsx*(nknot+3): eig will contain test eigenvalues (output)
c x needs to have ncx+nsx*4 columns, to include all spline basis functions
c if smoothing parameters are input need to be in
c components nsx*2+1 to nsx*3 of dsub
c on output x(.,ncx+1) to x(.,ncx+2*nsx) will be overwritten with
c estimates of spline terms and variances
c dsub 3*nsx+1,... will have inverse 2nd deriv matrix
c
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of penm's
c isub(5) reserved for the beginning index (in dsub) of unpenalized inf
c isub(6+1) to isub(6+nsx) gives col #'s (in X) of spline covs.
```

```
c isub(6+1+nsx) will be the beginning index of inter.
c
c dsub(1:nsx) are target df.
c dsub(nsx+1:2*nsx) will be attained df.
c dsub(2*nsx+1:3*nsx) will be smoothing parameters
c
c isub must have length at least 6+nsx+nsx*no
c
c    calls sph1, calvar, tstcv, cestc directly
c  indirect calls: coxrg, coxft, coxg, strat, ut, uft, tint, tint4
c   cholg, solve sortg dperm sub8, sub8c, sub9c, eval8, act8, extrct,
c   matop5, adisw, intsb2, pencb, spln, penh, gdf, degf2, penlik, eigen,
c   qfg, pdi, pdi2
      no=nov(1)
      nrx=nov(2)
      ncx=nov(3)
      istr=nov(4)
      nknot=nov(12)
      mit=nov(5)
      knopt=nov(6)
      ksmopt=nov(7)
      nfx=nov(8)
      nsx=nov(9)
      nest=nov(13)
      nk2=nknot+2
      nk3=nknot+3
      nk4=nknot+4
      np=nfx+nsx*nk2
      nkk=max(np-nk2,nk2)
      kmd=(np-nk2)*(np-nk2)
      kmd=max(kmd,nk3*nk3+2*nk2*nkk)
c
c in subsequent routines dsub is divided into dsub & work.
c the dsub part must be 3*nsx (for dft,dfa,alpha)
c                     + 4*(nknot+4)*nsx (penalty matrices)
c                     + np*np (unpen inf)
c  + either nk3*nk3 (ksmopt<=0) or kmd (ksmopt>0)
c the work part (indexed beginning with mdw) needs to be
c  at least max(no,4*np+np*np in the call to coxrg
c in the call to tstcv the total length of dsub must be at least
c 3*nsx+np*np+(nk3*nk3)*3
c
      if (ksmopt.gt.0) then
        mdw=3*nsx+np*np+4*nk4*nsx+kmd+1
      else
        mdw=3*nsx+np*np+4*nk4*nsx+nk3*nk3+1
      endif
      call sph1 (s,c,x,istr,isub,tknot,times,
     &beta,lik0,lik(1),sb,vv,iwork,dsub(mdw),dsub(1),eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1,z2,gn)
      if (icnv.gt.0) then
        vv(1)=-icnv
        return
      endif
      lik(2)=uplik
      if (nsx.gt.0) then
        mdw2=3*nsx+1+np*np
        mdw3=mdw2+nk3*nk3
        mdw4=mdw3+nk3*nk3
c note the call to tstcv overwrites some of the index info
c in iwork
        if (nov(10).gt.0) then
          call calvar(vv,np,np,dsub(isub(5)),np,vv,np,dsub(mdw))
```

```
c copy inverse 2nd deriv matrix to dsub(3*nsx+1),...
         m2=3*nsx+1
         do 21 i=0,np*np-1
  21     dsub(m2+i)=dsub(mdw+i)
        if (nov(10).gt.1) then
           call tstcv(np,nk2,nfx,nsx,beta,vv,np,dsub(m2),np,
     &       dsub(mdw2),dsub(mdw3),testr,iwork,dsub(mdw4),isub(7),eig)
         endif
        endif
        call cestc(x,nrx,ncx,no,nfx,nsx,isub(7),beta,vv,
     &    np,np,nknot,tknot,nov(12),nest)
       endif
       nov(12)=nknot
       return
       end
c routine sph1 for fitting proportional hazards spline models
c called by sph
c file contains sph1, sub8, sub8c, sub9c, eval8, act8
c all but sph1 are called by coxrg as part of fitting the model
c
c  routines in this file call coxrg, strat, sortg, extrct, cholg, matop5,
c    adisw, intsb2, pencb, spln, penh, gdf, penlik.
c
       subroutine sph1(s,c,x,istr,isub,tknot,
     &times,beta,lik0,lik,sb,vv,iwork,work,dsub,eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1,z2,gn)
       double precision vv(1),sb(1),work(1),beta(1),times(1),s(no)
       double precision dsub(1),tknot(-2:nk3,nsx)
       double precision c(no),x(nrx,ncx),eps,lik,lik0
C*************************************************************
       double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C*************************************************************
C>>>>>
       double precision wp(1),ts1p(1),s1p(1),z2(no,np)
       integer id(1),gn(1)
C<<<<<
       integer iwork(1),isub(1)
       external eval8,act8,sub8
       common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
      &nk2,nk3,nk4,malph
c
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of penm's
c isub(5) reserved for the beginning index (in dsub) of unpenalized inf
c isub(6+1) to isub(6+nsx) gives col #'s (in X) of spline covs.
c isub(6+1+nsx) will be the beginning index of inter.
c
c dsub(1:nsx) are target df.
c dsub(nsx+1:2*nsx) will be attained df.
c dsub(2*nsx+1:3*nsx) will be smoothing parameters
c
c  X must have ncx+4*nsx columns although when the call is made
c
c isub must have length at least 6+nsx+nsx*no
c
c  first get knot locations:
c
c this repeats some code from coxrg, but need to determine this first
c so splines can be calculated before calling coxrg.
c
       mnstr=1
       miuti=mnstr+1
       mtduti=miuti+2*no
```

```
c miwk needs length max(no,np)
      miwk=mtduti+no
      mnostr=miwk+max(no,np)
      call strat(istr,s,c,x,iwork(mnostr),nstr,iwork(miwk),work,
     &no,ncx,nrx,iwork(miuti),iwork(mtduti))
c      write (6,*) nstr
c      call pr1(iwork(miuti),iwork(mtduti),no)
      iwork(mnstr)=nstr
      mnuti=mnostr+nstr
c proportional hazards, so length of itud is 2*nstr
      ntud=1
      mitud=mnuti+nstr
c      write (6,*) (iwork(i),i=mnostr,mnostr+nstr-1)
c      write (6,*) (iwork(i),i=mnuti,mnuti+nstr-1)
      malph=2*nsx+1
      if (ksmopt.ge.0) then
      do 749 i=0,nsx-1
 749    dsub(malph+i)=1
      endif
      isub(3)=7+nsx
      isub(4)=malph+nsx
      isub(5)=isub(4)+4*nk4*nsx
c      write (6,*) nknot,no
c calc knot locations and penalty matrices
  89  nknp=nknot
      l1=isub(4)
      if (nsx.gt.0) then
      do 23 k=1,nsx
      if (knopt.le.0) then
        do 5 i=1,no
        work(i)=x(i,isub(6+k))
   5    continue
        call sortg(work,iwork(miwk),1,no)
        call intsb2(nknot,work,tknot(0,k),no)
      do 6 i=1,2
      tknot(-i,k)=tknot(0,k)
   6    tknot(nknot+i+1,k)=tknot(nknot+1,k)
      endif
      call pencb(nknot,tknot(-2,k),dsub(l1),nk4)
      l1=l1+4*nk4
  23  continue
      if (nknot.ne.nknp) then
      nk2=nknot+2
      nk3=nknot+3
      nk4=nknot+4
      go to 89
      endif
c  calc splines
      do 25 k=1,nsx
      l1=isub(3)+(k-1)*no-1
c last +1 in l2 is to allow for the strata variable
      l2=ncx+4*(k-1)
      l3=isub(6+k)
c      do 35 i = 1, no
c         do 30 j = 1, np
c            z2(i,k) = x(i,k)
c  30     continue
c  35  continue
      do 26 i=1,no
      call spln(nknot,3,tknot(-2,k),x(i,l3),1,isub(l1+i),work,1)
      do 27 ii=1,4
  27  x(i,l2+ii)=work(ii)
  26  continue
```

```
 25  continue
     endif
     np=nfx+nsx*nk2
     ntime=0
     do 35 i = 1, no
        do 30 j = 1, np
           z2(i,j) = x(i,j)
 30     continue
 35  continue
     do 28 i=1,np
 28  beta(i)=0
     call coxrg(s,c,x,no,nrx,ncx,istr,np,mit,1,ntime,times,beta,
    &lik0,lik,sb,vv,iwork,work,act8,eval8,sub8,isub,dsub,eps,icnv,
    &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1,gn)
     return
     end

     subroutine eval8(g,gp,b,lci,i,l,s,c,xx,nrx2,ncx2,no2,np2,lc2,nact,
    &iact,isub,dsub)
     double precision gp(np),s(no),c(no),xx(nrx,ncx),dsub(1),b(np),g
     integer isub(1),iact(1)
     common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
    &nk2,nk3,nk4,malph
     do 9 ii=1,np
  9  gp(ii)=0
     g=0
     do 10 ii=1,nfx
       gp(ii)=xx(i,ii)
 10  g=g+gp(ii)*b(ii)
     do 15 ii=1,nsx
       l1=ncx+4*(ii-1)
       l2=isub(3)+(ii-1)*no+i-1
       if (isub(l2).eq.0) go to 15
       l3=nfx+nk2*(ii-1)+isub(l2)-1
       l4=min(nk2,isub(l2)+3)-isub(l2)+1
       do 11 iq=1,l4
       gp(l3+iq)=xx(i,l1+iq)
 11    g=g+gp(l3+iq)*b(l3+iq)
 15  continue
     return
     end

     subroutine eval8n(zc,lci,i,l,s,c,xx,nrx2,ncx2,no2,np2,lc2,nact,
    &iact,isub,dsub,nfx1)
     double precision s(no),c(no),xx(nrx,ncx),dsub(1)
     integer isub(1),iact(1),zc(ncx,3)
     common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
    &nk2,nk3,nk4,malph
     nfx1 = nfx
     do 15 ii=1,nsx
       l1=ncx+4*(ii-1)
       l2=isub(3)+(ii-1)*no+i-1
       if (isub(l2).eq.0) go to 15
       l3=nfx+nk2*(ii-1)+isub(l2)-1
       l4=min(nk2,isub(l2)+3)-isub(l2)+1
       zc(ii,1) = l1
       zc(ii,2) = l3
       zc(ii,3) = l4
 15  continue
     return
     end
```

```
      subroutine act8(lci,l1,l2,iuti,no2,s,c,x,nrx2,ncx2,iact,nact,
     &np2,lc2,isub,dsub)
      double precision s(no),c(no),x(nrx,ncx),dsub(1)
      integer iuti(no,2),iact(1),isub(1)
      common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph
      do 10 i=1,np
 10   iact(i)=i
      nact=np
      return
      end

      subroutine sub9c(beta,lik,sb,vv,penm,v3,dft,dfa,alpha,v6,vvi)
      double precision beta(np),lik,sb(np),vv(np,np),v3(np,np)
      double precision penm(nk4,4,nsx),dft(nsx),dfa(nsx),alpha(nsx)
      double precision v6(1),mindf,vvi(np,np)
      common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph
c first update values of smoothing parameters to give required df.
      mv8=1+nk2*nk2
      np2=np-nk2
      nkk=max(np2,nk2)
      mv9=mv8+nk2*nkk
      do 820 i=1,np
      do 820 j=1,np
 820  vvi(i,j)=vv(i,j)
      do 821 j=2,nsx
      l1=nfx+1+(j-1)*nk2
      call penh(nk2,penm(1,1,j),nk4,4,vvi,l1,np,alpha(j))
 821  continue
      do 819 j=1,nsx
      l1=nfx+1+(j-1)*nk2
      l2=l1+nk2-1
      call extrct(vvi,np,np,l1-1,l2+1,v6,np2)
      call cholg(np2,v6,v3,np2,np2)
      call matop5(vv,np,v3,np2,l1,l2,v6(mv8),nk2,v6(mv9),v6,nkk)
      dfa(j)=dft(j)
c  3rd from last arg in gdf # dcols used
c in pen matrix, 4th from last mindf Here this is 0 since const and lin
c terms modelled separately
      mindf=0.d0
      call gdf(nk2,penm(1,1,j),v6(mv9),v6(mv8),v6,nkk,dfa(j),
     &alpha(j),v3,mindf,4,nk4,iopp1)
c
c  if alg failed then return without applying penalty
c
      if (iopp1.gt.0) return
c      write (6,100) dft(j),dfa(j),alpha(j)
      call penh(nk2,penm(1,1,j),nk4,4,vvi,l1,np,alpha(j))
      if (j.lt.nsx) then
        call penh(nk2,penm(1,1,j+1),nk4,4,vvi,l1+nk2,np,-alpha(j+1))
      endif
 819  continue
c then copy vv to v3, vvi to vv:
      do 10 i=1,np
      do 10 j=1,np
      v3(i,j)=vv(i,j)
 10   vv(i,j)=vvi(i,j)
c then add penalty terms to lik,score, (lik,sb)
      do 535 j=1,nsx
      l1=nfx+1+(j-1)*nk2
      call penlik(nk2,beta(l1),penm(1,1,j),nk4,4,lik,sb(l1),alpha(j))
```

```
c       call penh(nk2,penm,nk4,4,vv,l1,np,alpha(j))
 535   continue
 100   format (3e15.8)
       return
       end

       subroutine sub8(np2,beta,lik,sb,vv,vvi,isub,dsub,nit)
       double precision beta(np),lik,sb(np),vv(np,np),dsub(1)
       double precision vvi(1),uplik
       integer isub(1)
       common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
      &nk2,nk3,nk4,malph
       common /up/ uplik
c to provide working matrices, dsub must be at least
c 3*nsx+4*(n or ntud,depending on ntime)+np*np+2*(nknot+4)**2
       mv6=isub(5)+np*np
        uplik=lik
c        call dblepr('lik',3,lik,1)
       if (nsx.gt.0) then
        if (ksmopt.gt.0.and.nsx.gt.1.and.nit.lt.9.and.nit.gt.0) then
       call sub9c(beta,lik,sb,vv,dsub(isub(4)),dsub(isub(5)),
      &dsub(1),dsub(1+nsx),dsub(malph),dsub(mv6),vvi)
        else
       call sub8c(beta,lik,sb,vv,dsub(isub(4)),dsub(isub(5)),
      &dsub(1),dsub(1+nsx),dsub(malph),dsub(mv6),vvi,nit)
c       write (6,*) lik
        endif
       endif
cc       call dblepr('dsub',4,dsub,3)
c       call dblepr('plik',4,lik,1)
 110   format (e20.12)
       return
       end

       subroutine sub8c(beta,lik,sb,vv,penm,v3,dft,dfa,
      &alpha,v6,v7,nit)
       double precision beta(np),lik,sb(np),vv(np,np),v3(np,np)
       double precision penm(nk4,4,nsx),dft(nsx),dfa(nsx),alpha(nsx)
       double precision v6(nk2,nk2),v7(nk2,nk2),mindf
       common /params/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
      &nk2,nk3,nk4,malph
c first update values of smoothing parameters to give required df.
c note that adisw does not change vv.
c ksmopt<0 means use input value of alpha
c ksmopt=0 means calc using crude alg on first iter only
       if ((nit.eq.1.and.ksmopt.eq.0).or.(ksmopt.gt.0.and.nit.lt.9.
      &and.nit.gt.0)) then
       do 819 j=1,nsx
       l1=nfx+1+(j-1)*nk2
       l2=l1+nk2-1
       call adisw(l1,l2,np,vv,v3,np)
       dfa(j)=dft(j)
c  3rd from last arg in gdf # dcols used
c in pen matrix, 4th from last mindf Here this is 0 since const and lin
c terms modelled separately
       mindf=0.d0
       call gdf(nk2,penm(1,1,j),v3,v3,v6,np,dfa(j),alpha(j),v7,mindf,
      &4,nk4,iopp1)
c
c  if alg failed then return without applying penalty
c
       if (iopp1.gt.0) return
c       write (6,100) dft(j),dfa(j),alpha(j)
```

```
 819  continue
      endif
c then copy vv to v3:
      do 10 i=1,np
      do 10 j=1,np
  10  v3(i,j)=vv(i,j)
c then add penalty terms to lik,score, and inf (lik,sb,vv)
      do 535 j=1,nsx
      l1=nfx+1+(j-1)*nk2
      call penlik(nk2,beta(l1),penm(1,1,j),nk4,4,lik,sb(l1),alpha(j))
      call penh(nk2,penm(1,1,j),nk4,4,vv,l1,np,alpha(j))
 535  continue
 100  format (3e15.8)
      return
      end

      subroutine tstcv(np,nk2,nfx,nsx,beta,vv,lvv,v4,lv4,v6,v7,test,
     &iwork,v8,isx,eig)
c for spline given as linear+(nknot+2) cubic B-spline terms
c calcs Wald tests of no association and linearity
c nfx is the total number of vars, nsx the number of spline terms.
c isx(j) gives which entry in beta corresponds to the linear part of the
c jth spline term.
c on input v4 is the inverse penalized 2nd derivative matrix, and vv the
c estimated var-cov matrix of the paramters estimates.
c in this routine iwork must be at least 2*(nk2+1)
c in length
c v6,v7,v8 are work space,
c on output eig will contain the eigenvalues and test the test
c results
c calls pdi2, sortg, eigen from ~/src/util.a & qfg from ~/src/dist.a
      double precision vv(lvv,np),beta(np),v4(lv4,np)
      double precision v6(nk2+1,nk2+1),v7(nk2+1,nk2+1),test(6,nsx)
      double precision v8(nk2+1,nk2+1),qfg,trace(7),eig(nk2+1,2*nsx)
      integer iwork(1),isx(nsx)
c  stats:
      nk3=nk2+1
      do 50 k=1,nsx
      kk=isx(k)
      l1=nfx+(k-1)*nk2
c  overall test for var i:
c  first calc stat:
      v6(1,1)=v4(kk,kk)
      v8(1,1)=beta(kk)
      do 15 i=1,nk2
      ii=i+1
      v6(1,ii)=v4(kk,l1+i)
      v6(ii,1)=v6(1,ii)
      v8(ii,1)=beta(l1+i)
      do 15 j=1,nk2
      v6(ii,j+1)=v4(i+l1,j+l1)
  15  continue
      call pdi2(nk3,v6,v7,nk3,nk3)
      test(1,k)=0
      do 27 ii=1,nk3
      iwork(ii)=1
      test(2,k)=0
      do 28 jj=1,ii
  28  test(2,k)=test(2,k)+v7(jj,ii)*v8(jj,1)
      test(1,k)=test(1,k)+test(2,k)*test(2,k)
  27  continue
c then get eigenvalues to calc dist
      v6(1,1)=vv(kk,kk)
```

```
      do 17 i=1,nk2
      ii=i+1
      v6(1,ii)=vv(kk,l1+i)
      v6(ii,1)=v6(1,ii)
      do 17 j=1,nk2
   17 v6(ii,j+1)=vv(i+l1,j+l1)
      do 18 i=1,nk3
      do 18 j=i,nk3
      v8(i,j)=0
      do 19 ii=1,i
      do 19 jj=1,j
   19 v8(i,j)=v8(i,j)+v7(ii,i)*v6(ii,jj)*v7(jj,j)
      v8(j,i)=v8(i,j)
   18 continue
      call eigen(v8,nk3,nk3,eig(1,2*k-1),v6,1,iflg)
      call sortg(eig(1,2*k-1),iwork,1,nk3)
      do 37 ii=1,nk3
      iwork(ii)=1
   37 v8(ii,1)=0
      test(2,k)=1-qfg(eig(1,2*k-1),v8,iwork,nk3,0.d0,test(1,k),5000,
     &1.d-5,iwork(nk3+1),trace,ifault)
      test(3,k)=0
      do 35 i=1,nk3
   35 test(3,k)=test(3,k)+eig(i,2*k-1)
c  test for linearity :
c  calc stat:
      do 215 i=1,nk2
      do 215 j=1,nk2
  215 v6(i,j)=v4(i+l1,j+l1)
c      call matwr(v6,nk2,nk3,88)
      call pdi2(nk2,v6,v7,nk3,nk3)
      test(4,k)=0
      do 427 ii=1,nk2
      iwork(ii)=1
      test(5,k)=0
      do 428 jj=1,ii
  428 test(5,k)=test(5,k)+v7(jj,ii)*beta(l1+jj)
      test(4,k)=test(4,k)+test(5,k)*test(5,k)
  427 continue
c get eigenvalues:
      do 317 i=1,nk2
      do 317 j=1,nk2
  317 v6(i,j)=vv(i+l1,j+l1)
c      call matwr(v6,nk2,nk3,89)
      do 318 i=1,nk2
      do 318 j=i,nk2
      v8(i,j)=0
      do 319 ii=1,i
      do 319 jj=1,j
  319 v8(i,j)=v8(i,j)+v7(ii,i)*v6(ii,jj)*v7(jj,j)
      v8(j,i)=v8(i,j)
  318 continue
c      call matwr(v8,nk2,nk3,90)
      call eigen(v8,nk2,nk3,eig(1,2*k),v6,1,iflg)
      call sortg(eig(1,2*k),iwork,1,nk2)
      do 327 ii=1,nk2
      iwork(ii)=1
  327 v8(ii,1)=0
      test(5,k)=1-qfg(eig(1,2*k),v8,iwork,nk2,0.d0,test(4,k),5000,
     &1.d-5,iwork(nk3+1),trace,ifault)
      test(6,k)=0
      do 335 i=1,nk2
  335 test(6,k)=test(6,k)+eig(i,2*k)
```

```
 50   continue
      return
      end

      subroutine cestc(x,nrx,ncx,no,nfx,nsx,isx,
     &beta,var,np,lvar,nknot,tknot,nknto,nest)
c routine for calculating function estimates and variances
c from output of sph/wsph.
c x is covariate values including spline terms.  for info on ordering
c etc see sph.f
      double precision x(nrx,1),est,v,c(9),w(4),bl,w2(4),q1,q2,q3
      double precision beta(np),var(lvar,np),tknot(-2:(nknto+3),nsx)
      integer inter,isx(1),ind(9)
      if (nest.gt.no) nest=no
      nk2=nknot+2
      do 5 k=1,nsx
      kk=isx(k)
      l1=nfx+(k-1)*nk2
c
c  baseline value based on knots:
c
c      bl=(tknot((nknot+1)/2,k)+tknot((nknot+2)/2,k))/2
c
c  or on means of covariates:
c
      bl=0
      do 7 ii=1,no
   7  bl=bl+x(ii,kk)
      bl=bl/no
c
      call spln(nknot,3,tknot(-2,k),bl,1,intbl,w,1)
      m2=min(intbl+3,nk2)-intbl+1
      do 201 jj=1,m2
      ind(jj)=intbl-1+jj+l1
 201  c(jj)=-w(jj)
      q1=bl
      q2=q1
      do 205 ii=1,no
      if (x(ii,kk).ge.tknot(0,k)) q1=min(x(ii,kk),q1)
 205  if (x(ii,kk).le.tknot(nknot+1,k)) q2=max(x(ii,kk),q2)
      q3=(q2-q1)*1.d-8
      q1=q1+q3
      q2=q2-q3
      do 50 i=1,nest
      q3=q1+(i-1)*(q2-q1)/(nest-1)
      call spln(nknot,3,tknot(-2,k),q3,1,inter,w2,1)
      if (inter.le.0) then
      x(i,kk)=-1
      x(i,ncx+2*k-1)=-1
      x(i,ncx+2*k)=-1
      go to 50
      endif
      l2=l1+inter-1
      l3=min(inter+3,nk2)-inter+1
      do 202 jj=1,l3
      ind(jj+m2)=l2+jj
 202  c(jj+m2)=w2(jj)
      lu=l3+m2+1
      ind(lu)=kk
      c(lu)=q3-bl
      est=0
      v=0
      do 10 ii=1,lu
```

```
      est=est+beta(ind(ii))*c(ii)
      do 11 jj=1,lu
   11 v=v+var(ind(ii),ind(jj))*c(ii)*c(jj)
   10 continue
      x(i,kk)=q3
      x(i,ncx+2*k-1)=est
      x(i,ncx+2*k)=v
   50 continue
    5 continue
      return
      end

c
c This software comes with absolutely no guarantees.  You have permission to
c use it for any noncommercial purpose, and to modify it as needed.
c Copyright 1992 by Robert Gray
c
      subroutine sphi(nov,s,c,x,tknot,beta,lik0,lik,sb,vv,
     &iwork,dsub,isub,testr,eig,eps,z,exz,s0,s1,ts1,w
     &wp,ts1p,id,s1p,z1)
      double precision s(1),c(1),x(1),tknot(1),times(1),beta(1)
      double precision lik(2),eps,sb(1),vv(1),dsub(1),testr(1),eig(1)
      double precision uplik,scx,ucx,scy,ucy,lik0
C*************************************************************
      double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C*************************************************************
C>>>>>
      double precision wp(1),ts1p(1),s1p(1)
      integer id(1)
C<<<<<
      integer nov(15),iwork(1),isub(1)
      common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
     &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
      common /up/ uplik
c
c includes nfx fixed covs and a 2-cov interaction modelled using
c tensor product B-splines
c on input isub(7) to isub(6+nfx) should contain the column numbers
c in x for the fixed covs. isub(1) and isub(2) should give the column
c numbers of the 1st and 2nd terms for the spline interaction
c (isx and isy)
c
c  nov parameters:
c   1=no, 2=nrx(=no), 3=ncx, 4=istr (col # of x for strata, -1 if none,
c   5=maxiter, 6=knot optiont (1 use knots provided, 0 program calculates),
c   7=smoothing option (<0 use input smoothing params, 0 calc smoothing
c   param only in first iteration, >0 recalc after each iteration)
c   8=nfx (nfx+nsx in sph, nfx is #col in linear.cov, nsx in spline.cov),
c   9=nsx, 10=analysis option (<=0 estimates only, 1 est & var, >1 est,
c   var & test),
c     sphi:  11,12=nknx,nkny, 13,14=nestx,nesty, 15=rescale opt (0=rescale
c     covariates, 0=do not)
c
c dsub must have length max(3+nex*ney*4+np*np,3+16*nkx4*nky4+nb*nb+np*np+
c max(max(no,4*np+np*np),np*np+nb*nb))
c iwork must have length at least 3*no+max(no,np)+1+2*nstr+2*nstr (since ntud=1)
c where nstr is the # strata.
c isub must have length 6+nfx+2*no
c testr must have length 12, (order will be stat,pv,df for overall
c test, ave effect of x, ave effect of y, no interaction test)
c length(eig)=4*(nkx4*nky4-1): eig will contain test eigenvalues (output)
c on input ncx should indicate the number of columns of x with input
c data (which should be consecutive in X.  columns ncx+1 to ncx+4 will
```

```
c be used for x bspline terms, and columns ncx+5 to ncx+8 for the y
c bspline terms.  Note that x therefore must have at least ncx+8 columns
c if smoothing parameter is input needs to be in component 1 of dsub
c target df for total 2 var interaction term needs to be in dsub(2)
c and attained df on output will be in dsub(3)
c  On output estimates will be in
c dsub(4), ... dsub(3+nex*ney*4), with the first nex*ney comps giving
c the x coords, then the y coords, then the estimates, then the variances
c total # parameters is nfx+(nknx+4)*(nkny+4)-1
c inverse 2nd deriv matrix will be in dsub(4+nex*ney)...
c
c total length of tknot must be nknx+6+nkny+6
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of penm's
c isub(5) reserved for the beginning index (in dsub) of unpenalized inf
c isub(6+1) to isub(6+nfx) gives col #'s (in X) of fixed covs.
c isub(6+1+nfx) will be the beginning index of inter.
c
c
c note that the parameter order loops over y within x: ie
c ordered b(x1,y1),b(x1,y2),...,b(x1,y[nky+4]),b(x2,y1),...
c
c    calls sphi1, calvar, testi, cesti directly
c
c  indirect calls: coxrg, coxft, coxg, strat, ut, uft, tint, tint4
c   cholg, solve sortg dperm subi, subic, evali, acti,
c   adisw, intsb2, pencxc, spln, penh2, gdf2, degf3, penlk2, eigen,
c   qfg, pdi, pdi2, penhg, penlkg, csplcf, int1, int2, st1, st2, intb1
c
      no=nov(1)
      nrx=nov(2)
      ncx=nov(3)
      istr=nov(4)
      isx=isub(1)
      isy=isub(2)
      nknx=nov(11)
      nkny=nov(12)
      mit=nov(5)
      knopt=nov(6)
      ksmopt=nov(7)
      nfx=nov(8)
      nkx2=nknx+2
      nkx3=nknx+3
      nkx4=nknx+4
      nky2=nkny+2
      nky3=nkny+3
      nky4=nkny+4
      np=nfx+nky4*nkx4-1
      nb=np-nfx
c
c in subsequent routines dsub is divided into dsub & work.
c the dsub part must be 3 (for alpha,dft,dfa)
c                     + 4*(nkx4)*4*(nky4) (penalty matrices)
c                     + np*np (unpen inf)
c                     + nb*nb (work matrix in dsub)
c the work part (indexed beginning with mdw) needs to be
c  at least max(no,4*np+np*np) in the call to coxrg
c  & at least np*np+nb*nb for the call to testi
c
c  also, to hold the output, the two parts together must be at least
c  3+nestx*nesty*4+np*np
c
      mdw=3+np*np+4*nkx4*4*nky4+nb*nb
```

```
      if (nov(15).le.0) then
        ucx=0
        ucy=0
        isx2=(isx-1)*nrx
        isy2=(isy-1)*nrx
        do 444 i=1,no
        ucx=ucx+x(i+isx2)
444     ucy=ucy+x(i+isy2)
        ucx=ucx/no
        ucy=ucy/no
        scx=0
        scy=0
        do 443 i=1,no
        scx=scx+(x(i+isx2)-ucx)**2
443     scy=scy+(x(i+isy2)-ucy)**2
        scx=sqrt(scx/(no-1))
        scy=sqrt(scy/(no-1))
        do 442 i=1,no
        x(i+isx2)=(x(i+isx2)-ucx)/scx
442     x(i+isy2)=(x(i+isy2)-ucy)/scy
      else
        ucx=0
        ucy=0
        scx=1
        scy=1
      endif
      call sphi1 (s,c,x,istr,isub,tknot,times,
     &beta,lik0,lik(1),sb,vv,iwork,dsub(mdw),dsub(1),eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      if (icnv.gt.0) then
         vv(1)=-icnv
         return
      endif
      lik(2)=uplik
      nb=np-nfx
      m6=4
      m7=m6+nb*nb
      m8=mdw+np*np
      if (nov(10).gt.0) then
        call calvar(vv,np,np,dsub(isub(5)),np,vv,np,dsub(mdw))
      endif
      if (nov(10).gt.1) then
      call testi(np,nb,nkx4,nky4,nfx,beta,vv,np,dsub(mdw),np,
     &dsub(m6),dsub(m7),testr,iwork,dsub(m8),eig,nknx,nknx,tknot)
      endif
c move the inverse 2nd deriv matrix from cal var to start at
c dsub(4+nex*ney):
      m2=4+4*nov(13)*nov(14)
      if (m2.gt.mdw) then
        do 20 i=np*np-1,0,-1
  20    dsub(m2+i)=dsub(mdw+i)
      else if (m2.lt.mdw) then
        do 22 i=0,np*np-1
  22    dsub(m2+i)=dsub(mdw+i)
      endif
      call cesti(nknx,nknx,tknot,np,beta,vv,np,nov(13),nov(14),dsub(4),
     &nfx,ucx,ucy,scx,scy)
      return
      end
c fits 2 dim tensor product bspline in prop hazards model
c  file contains sphi1 acti evali subi subic gdf2 degf3
c  acti evali subi subic are called by coxrg as part of fitting model
c
```

```
c  in addition, routines in this file call strat, sortg, coxrg,
c  cholg, adisw, solve, intsb2, pencxc, spln, penh2, penlk2
c
      subroutine sphi1(s,c,x,istr,isub,tknot,
     &times,beta,lik0,lik,sb,vv,iwork,work,dsub,eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      double precision vv(1),sb(1),work(1),beta(1),times(1),s(no)
      double precision dsub(1),tknot(-2:nkx3,2)
      double precision c(no),x(nrx,ncx),eps,lik,lik0
C*************************************************************
      double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C*************************************************************
C>>>>>
      double precision wp(1),ts1p(1),s1p(1)
      integer id(1)
C<<<<<
      integer iwork(1),isub(1)
      external evali,acti,subi
      common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
     &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
c total length of tknot must be nknx+6+nkny+6
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of penm's
c isub(5) reserved for the beginning index (in dsub) of unpenalized inf
c isub(6+1) to isub(6+nfx) gives col #'s (in X) of fixed covs.
c isub(6+1+nfx) will be the beginning index of inter.
c
c  X must have ncx+8 columns although when the call is made
c
c isub must have length at least 7+nfx+2*no
c
c note that the parameter order loops over y within x: ie
c ordered b(x1,y1),b(x1,y2),...,b(x1,y[nky+4]),b(x2,y1),...
c
c  first get knot locations:
c
c this repeats some code from coxrg, but need to determine this first
c so splines can be calculated before calling coxrg.
c
      mnstr=1
      miuti=mnstr+1
      mtduti=miuti+2*no
c miwk needs length max(no,np)
      miwk=mtduti+no
      mnostr=miwk+max(no,np)
      call strat(istr,s,c,x,iwork(mnostr),nstr,iwork(miwk),work,
     &no,ncx,nrx,iwork(miuti),iwork(mtduti))
c      write (6,*) nstr
      iwork(mnstr)=nstr
      mnuti=mnostr+nstr
c proportional hazards, so length of itud is 2*nstr
      ntud=1
      mitud=mnuti+nstr
c      write (6,*) (iwork(i),i=mnostr,mnostr+nstr-1)
c      write (6,*) (iwork(i),i=mnuti,mnuti+nstr-1)
      isub(3)=7+nfx
      isub(4)=4
      isub(5)=isub(4)+nkx4*nky4*16
c      write (6,*) no,nknx,nkny,isx,isy
c calc knot locations and penalty matrices
  89  nknpx=nknx
      nknpy=nkny
      continue
```

```
      if (knopt.le.0) then
        do 5 i=1,no
        work(i)=x(i,isx)
    5   continue
        call sortg(work,iwork(miwk),1,no)
        call intsb2(nknx,work,tknot(0,1),no)
        do 7 i=1,2
        tknot(-i,1)=tknot(0,1)
    7   tknot(nknx+i+1,1)=tknot(nknx+1,1)
        do 6 i=1,no
        work(i)=x(i,isy)
    6   continue
        call sortg(work,iwork(miwk),1,no)
        call intsb2(nkny,work,tknot(0,2),no)
        do 8 i=1,2
        tknot(-i,2)=tknot(0,2)
    8   tknot(nkny+i+1,2)=tknot(nkny+1,2)
      endif
c      write (6,*) ((tknot(i,j),i=-2,nky3),j=1,2)
c in the next call the last 2 arguments require 36*nkx4 &36*nky4 room
      call pencxc(nknx,tknot(-2,1),nkny,tknot(-2,2),dsub(isub(4)),
     &dsub(isub(5)),dsub(isub(5)+12*nkx4))
      if (nknx.ne.nknpx.or.nknpy.ne.nkny) then
      nkx2=nknx+2
      nkx3=nknx+3
      nkx4=nknx+4
      nky2=nkny+2
      nky3=nkny+3
      nky4=nkny+4
      go to 89
      endif
 102  format (4e15.6)
c  calc splines
c last +1 in 12 is to allow for the strata variable
      l1=isub(3)-1
      do 26 i=1,no
      call spln(nknx,3,tknot(-2,1),x(i,isx),1,isub(l1+i),work,1)
      do 27 ii=1,4
  27  x(i,ncx+ii)=work(ii)
  26  continue
      l1=isub(3)+no-1
      do 28 i=1,no
      call spln(nkny,3,tknot(-2,2),x(i,isy),1,isub(l1+i),work,1)
      do 29 ii=1,4
  29  x(i,ncx+4+ii)=work(ii)
  28  continue
c      do 500 i=1,no
c 500  write (45,101) isub(isub(3)+i-1),isub(isub(3)+no+i-1),
c     &(x(i,j),j=1,ncx+8)
c 101  format(2i4,12e12.4)
      np=nfx+nkx4*nky4-1
c      write (6,*) np,mit
c  initialize smoothing parameter for first iteration
      if (ksmopt.ge.0) dsub(1)=1.d0
      ntime=0
      do 38 i=1,np
  38  beta(i)=0
      call coxrg(s,c,x,no,nrx,ncx,istr,np,mit,1,ntime,times,beta,
     &lik0,lik,sb,vv,iwork,work,acti,evali,subi,isub,dsub,eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      return
      end
```

```
      subroutine evali(g,gp,b,lci,i,l,s,c,xx,nrx2,ncx2,no2,np2,lc2,nact,
     &iact,isub,dsub)
      double precision gp(np),s(no),c(no),xx(nrx,ncx),dsub(1),b(np),g
      integer isub(1),iact(1)
      common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
     &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
      do 9 ii=1,np
    9 gp(ii)=0
      g=0
      do 10 ii=1,nfx
        gp(ii)=xx(i,isub(6+ii))
        g=g+gp(ii)*b(ii)
   10 continue
c  first get which interval x and y coords are in:
      lx=isub(isub(3)+i-1)
      ly=isub(isub(3)+no+i-1)
      if (lx.le.0.or.ly.le.0) return
      ly2=ncx+4
      do 15 ii=lx,lx+3
        lx2=ii-lx+1+ncx
c 11 is one before first index of y terms for the iith x term:
        l1=nfx+nky4*(ii-1)+ly-1
        if (ii.eq.nkx4.and.ly+3.eq.nky4) then
          l2=3
        else
          l2=4
        endif
c        write (44,*) i,lx,ly,ii,l1,lx2,ly2
        do 11 iq=1,l2
          gp(l1+iq)=xx(i,lx2)*xx(i,ly2+iq)
          g=g+gp(l1+iq)*b(l1+iq)
   11   continue
   15 continue
      return
      end

      subroutine acti(lci,l1,l2,iuti,no2,s,c,x,nrx2,ncx2,iact,nact,
     &np2,lc2,isub,dsub)
      double precision s(no),c(no),x(nrx,ncx),dsub(1)
      integer iuti(no,2),iact(1),isub(1)
      common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
     &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
      do 10 i=1,np
   10 iact(i)=i
      nact=np
      return
      end

      subroutine subi(np2,beta,lik,sb,vv,vvi,isub,dsub,nit)
      double precision beta(np),lik,sb(np),vv(np,np),dsub(1)
      double precision vvi(1),uplik
      integer isub(1)
      common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
     &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
      common /up/ uplik
c to provide working matrices, dsub must be at least ?? (following
c  line is wrong)
c 3*nsx+4*(n or ntud,depending on ntime)+np*np+2*(nknot+4)**2
      nb=np-nfx
      mv6=isub(5)+np*np
      mv7=mv6+nb*nb
c      write (6,*) lik
      uplik=lik
```

```
c      call dblepr("lik",3,lik,1)
       call subic(beta,lik,sb,vv,dsub(isub(4)),dsub(isub(5)),
      &dsub(2),dsub(3),dsub(1),dsub(mv6),vvi,nit,nb)
c      write (6,*) lik
c      call dblepr("plik",4,lik,1)
 110   format (e20.12)
       return
       end

       subroutine subic(beta,lik,sb,vv,penm,v3,dft,dfa,
      &alpha,v6,v7,nit,nb)
       double precision beta(np),lik,sb(np),vv(np,np),v3(np,np)
       double precision penm(nky4,4,nkx4,4),dft,dfa,alpha
       double precision v6(nb,nb),v7(nb,nb),mindf
       common /parami/ no,nrx,ncx,isx,isy,nknx,nkny,mit,knopt,ksmopt,
      &np,nfx,nkx2,nkx3,nkx4,nky2,nky3,nky4
c ksmopt<0 means use input value of alpha
c ksmopt=0 means calc using crude alg on first iter only
       if ((nit.eq.1.and.ksmopt.eq.0).or.(ksmopt.gt.0.and.nit.lt.9.
      &and.nit.gt.0)) then
c first update values of smoothing parameters to give required df.
c note that adisw does not change vv.
       call adisw(nfx+1,np,np,vv,v3,np)
       dfa=dft
c  3rd from last arg in gdf # dcols used
c in pen matrix, 4th from last mindf Here I guess this is 2 since as
c alpha->infty, should get lin(x)+lin(y)
       mindf=2.d0
       call gdf2(nkx4,nky4,nb,penm,v3,v3,v6,np,dfa,alpha,v7,mindf,
      &nkx4,4,nky4,4,iopp1)
c if algorithm failed in gdf return without penalizing.
       if (iopp1.gt.0) return
c      write (6,100) dft,dfa,alpha
 819   continue
       endif
c then copy vv to v3:
       do 10 i=1,np
       do 10 j=1,np
  10   v3(i,j)=vv(i,j)
c then add penalty terms to lik,score, and inf (lik,sb,vv)
       l1=nfx+1
       call penlk2(nkx4,nky4,nb,beta(l1),penm,nkx4,4,nky4,4,lik,
      &sb(l1),alpha)
       call penh2(nkx4,nky4,nb,penm,nkx4,nky4,4,4,vv,l1,np,alpha)
 100   format (3e15.8)
       return
       end

       subroutine gdf2(nbx,nby,nbt,vv,v3,var,vvi,lb,df,alpha,v4,mindf,
      &mrx,mdx,mry,mdy,iopt)
c change 3-21-91:  initialize alpha before calling
c iopt not currently used
c on output,iopt =0 means algorithm worked ok
c iopt=1 means singular or other problems
c the fast option (iopt>0) only gives valid results if var and v3 are the same.
c nbt is the total # parameters,
c nbx the number of x basis functions, nby the number of y basis
c functions
c vv is the penalty matrix , v3 is the unpenalized -2nd deriv,
c var is the actual var matrix (both have row dim lb).  var and v3 can be
c the same matrix (vv v3 and var are not changed)
c vvi and v4 are working matrices with dim ntud3*ntud3.
c df is the target df (input) on output df=attained df,
```

```
c alpha is the smp (output)
c mindf is the minimum degree of freedom possible for this spline/pen
c combination.  (note that this is double precision)
c md is the number of nonzero diagonals in the penalty matrix
c mr the actual row dim of pen mat
c calls cholg, solve,(and degf3) and penh2.
      double precision vv(mry,mdy,mrx,mdx),vvi(nbt,nbt),v3(lb,lb)
      double precision al,au,ap,del,var(lb,lb),alpha,df
      double precision fl,fu,fp,v4(nbt,nbt),mindf,eps
      eps=1.d-3
      mit=100
      nit=0
      if (df.ge.nbt) then
        alpha=0
        df=nbt
        return
      else if (df.le.mindf) then
        df=mindf+.05d0
      endif
c
c calculate smoothing parameter to get appropriate degrees of freedom
c  start nonpd algorithm:
 200  iopt=0
      ap=alpha
      fu=df+1
 202  call degf3(nbx,nby,nbt,vv,v3,var,v4,vvi,ap,lb,fp,
     &mrx,mdx,mry,mdy)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      nit=nit+1
      if (abs(fp-df).lt.eps) go to 250
      if (nit.gt.mit) then
         iopt=1
         go to 250
      endif
      if (fp.gt.df) then
        al=ap
        fl=fp
        go to 205
      else
        au=ap
        fu=fp
        ap=ap/4
        go to 202
      endif
 205  if (fu.lt.df) go to 208
      au=al
      del=4
 206  au=au*del
      call degf3(nbx,nby,nbt,vv,v3,var,v4,vvi,au,lb,fu,
     &mrx,mdx,mry,mdy)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      nit=nit+1
      if (abs(fu-df).lt.eps) then
        ap=au
        fp=fu
        go to 250
      endif
```

```
      if (nit.gt.mit) then
         iopt=1
         go to 250
      endif
      if (fu.gt.df) then
        al=au
        fl=fu
        go to 206
      endif
  208 ap=(al+au)/2
      call degf3(nbx,nby,nbt,vv,v3,var,v4,vvi,ap,lb,fp,
     &mrx,mdx,mry,mdy)
      if (v4(1,1).lt.0) then
         iopt=1
         return
      endif
      if (abs(fp-df).lt.eps) go to 250
      nit=nit+1
      if (nit.gt.mit) then
          iopt=1
          go to 250
      endif
      if (fp.gt.df) then
        al=ap
        fl=fp
      else
        au=ap
        fu=fp
      endif
      go to 208
  250 alpha=ap
      df=fp
  110 format(4e18.10)
c      write (70,*) nit
      return
      end

      subroutine degf3(nbx,nby,nbt,vv,v3,var,v4,vvi,a0,lb,t1,
     &mrx,mdx,mry,mdy)
c  on output t1 is the trace (=df)
c  vv is pen v3 is inf v4 vvi are work var is the true var
      double precision vv(mry,mdy,mrx,mdx),v3(lb,lb),v4(nbt,nbt),a0
      double precision vvi(nbt),t1,var(lb,lb)
      do 10 i=1,nbt
      do 10 j=1,nbt
      v4(i,j)=v3(i,j)
   10 continue
      call penh2(nbx,nby,nbt,vv,mrx,mry,mdx,mdy,v4,1,nbt,a0)
      call cholg(nbt,v4,v4,nbt,nbt)
      if (v4(1,1).lt.0) then
      call dblepr("not pd in degf",14,v4(1,1),1)
  143  format ('matrix not pd in degf')
       endif
      t1=0
      do 20 i=1,nbt
      call solve(nbt,v4,var(1,i),vvi,nbt)
      t1=t1+vvi(i)
   20 continue
      return
      end

      subroutine testi(np,nb,nx,ny,nfx,beta,vv,lvv,v4,lv4,v6,v7,test,
     &iwork,v8,eig,nknx,nkny,tknot)
```

```
c For a tensor product b-spline computes test (Wald) that
c all coefs=0
c nb is the number of basis fcns for spline terms assumed to be
c components nfx+1 to nfx+nb of beta and var.
c on input v4 is the inverse penalized 2nd derivative matrix, and vv the
c estimated var-cov matrix of the paramters estimates.
c  length(iwork)=2*nk4
c v6,v7,v8 are work space,
c on output eig will contain the eigenvalues and test the test
c results
c someday will add tests of no interaction, and tests for average main
c effects
c calls pdi2, sortg, eigen from ~/src/util.a & qfg from ~/src/dist.a
c also intb1
      double precision vv(lvv,np),beta(np),v4(lv4,np)
      double precision v6(nb,nb),v7(nb,nb),test(3,4)
      double precision v8(nb,nb),eig(nb,4),tknot(-2:nknx+3,2)
      integer iwork(1)
c first overall test for an effect:
c  first calc stat:
      do 15 i=1,nb
      do 15 j=1,nb
      v6(i,j)=v4(i+nfx,j+nfx)
  15  continue
      call st1(nb,beta(nfx+1),v6,nb,v7,test(1,1))
c then get eigenvalues to calc dist
      do 17 i=1,nb
      do 17 j=1,nb
  17  v6(i,j)=vv(i+nfx,j+nfx)
      call st2(nb,v6,v7,v8,nb,test,eig,iwork)
c  test for main effects of x (weighted sum over y):
c test statistic:
c (any line numbers 2xx)
c col 1 of v8 will be new params:
c get weights(integrate over y):
      call intb1(nkny,tknot(-2,2),v8(1,2))
      ny2=nb-ny*(nx-1)
      do 201 i=1,nx
      v8(i,1)=0
      lx=nfx+(i-1)*ny
      nyu=ny
      if (i.eq.nx) nyu=ny2
      do 202 k=1,nyu
 202  v8(i,1)=v8(i,1)+v8(k,2)*beta(lx+k)
      do 203 j=i,nx
      nyu2=ny
      if (j.eq.nx) nyu2=ny2
      v7(i,j)=0
      lx2=nfx+(j-1)*nx
      do 204 ii=1,nyu
      do 204 jj=1,nyu2
 204  v7(i,j)=v7(i,j)+v8(ii,2)*v8(jj,2)*v4(ii+lx,jj+lx2)
      v7(j,i)=v7(i,j)
 203  continue
 201  continue
c now take contrasts:
      do 221 i=1,nx-1
      v8(i,1)=v8(i,1)-v8(i+1,1)
      do 221 j=i,nx-1
      v6(i,j)=v7(i,j)-v7(i+1,j)-v7(i,j+1)+v7(i+1,j+1)
      v6(j,i)=v6(i,j)
 221  continue
      call st1(nx-1,v8(1,1),v6,nb,v7,test(1,2))
```

```
c don't overwrite v7, because you need the same matrix in the call to
c st2
c get var-cov matrix:
      do 211 i=1,nx
      lx=nfx+(i-1)*ny
      nyu=ny
      if (i.eq.nx) nyu=ny2
      do 213 j=i,nx
      nyu2=ny
      if (j.eq.nx) nyu2=ny2
      v6(i,j)=0
      lx2=nfx+(j-1)*nx
      do 214 ii=1,nyu
      do 214 jj=1,nyu2
 214  v6(i,j)=v6(i,j)+v8(ii,2)*v8(jj,2)*vv(ii+lx,jj+lx2)
      v6(j,i)=v6(i,j)
 213  continue
 211  continue
c note that the params and weights in v8 are no longer needed:
      do 222 i=1,nx-1
      do 222 j=i,nx-1
      v8(i,j)=v6(i,j)-v6(i+1,j)-v6(i,j+1)+v6(i+1,j+1)
      v8(j,i)=v8(i,j)
 222  continue
      call st2(nx-1,v8,v7,v6,nb,test(1,2),eig(1,2),iwork)
c average y-effect
c (3xx)
c first get weights integrating over x
      call intb1(nknx,tknot(-2,1),v8(1,2))
      nx2=nb-nx*(ny-1)
      do 301 i=1,ny
      v8(i,1)=0
      ly=nfx+i
      nxu=nx
      if (i.eq.ny) nxu=nx2
      do 302 k=0,nxu-1
 302  v8(i,1)=v8(i,1)+v8(k+1,2)*beta(ly+k*ny)
      do 303 j=i,ny
      nxu2=nx
      if (j.eq.ny) nxu2=nx2
      v7(i,j)=0
      ly2=nfx+j
      do 304 ii=0,nxu-1
      do 304 jj=0,nxu2-1
 304  v7(i,j)=v7(i,j)+v8(ii+1,2)*v8(jj+1,2)*v4(ii*ny+ly,jj*ny+ly2)
      v7(j,i)=v7(i,j)
 303  continue
 301  continue
      do 321 i=1,ny-1
      v8(i,1)=v8(i,1)-v8(i+1,1)
      do 321 j=i,ny-1
      v6(i,j)=v7(i,j)-v7(i+1,j)-v7(i,j+1)+v7(i+1,j+1)
      v6(j,i)=v6(i,j)
 321  continue
      call st1(ny-1,v8(1,1),v6,nb,v7,test(1,3))
c remember, v7 needed later
c get var-cov matrix:
      do 311 i=1,ny
      ly=nfx+i
      nxu=nx
      if (i.eq.ny) nxu=nx2
      do 313 j=i,ny
      nxu2=nx
```

```
      if (j.eq.ny) nxu2=nx2
      v6(i,j)=0
      ly2=nfx+j
      do 314 ii=0,nxu-1
      do 314 jj=0,nxu2-1
 314  v6(i,j)=v6(i,j)+v8(ii+1,2)*v8(jj+1,2)*vv(ii*ny+ly,jj*ny+ly2)
      v6(j,i)=v6(i,j)
 313  continue
 311  continue
      do 322 i=1,ny-1
      do 322 j=i,ny-1
      v8(i,j)=v6(i,j)-v6(i+1,j)-v6(i,j+1)+v6(i+1,j+1)
      v8(j,i)=v8(i,j)
 322  continue
      call st2(ny-1,v8,v7,v6,nb,test(1,3),eig(1,3),iwork)
c test for interaction
c (4xx)
c 1. y contrasts:
c note: this all assumes that nb=nx*ny-1 (everything but the const
c in splines)
      ny2=ny-1
      l=0
      do 401 i=1,nx
      lx=nfx+(i-1)*ny
      nyu=ny-1
      if (i.eq.nx) nyu=ny2-1
      do 402 j=1,nyu
      l=l+1
      v8(l,1)=beta(lx+j)-beta(lx+j+1)
      l2=l-1
      do 403 i2=i,nx
      lx2=nfx+(i2-1)*ny
      nyu2=ny-1
      if (i2.eq.nx) nyu2=ny2-1
      jk=1
      if (i2.eq.i) jk=j
      do 403 j2=jk,nyu2
      l2=l2+1
      v7(l,l2)=v4(lx+j,lx2+j2)-v4(lx+j+1,lx2+j2)-v4(lx+j,lx2+j2+1)+
     &v4(lx+j+1,lx2+j2+1)
      v7(l2,l)=v7(l,l2)
 403  continue
      l2=l2+1
      v7(l,l2)=v4(lx+j,nfx+nb)-v4(lx+j+1,nfx+nb)
      v7(l2,l)=v7(l,l2)
 402  continue
 401  continue
      l=l+1
      v8(l,1)=beta(nfx+nb)
      v7(l,l)=v4(nfx+nb,nfx+nb)
c x-contrasts of y contrasts:
      l=0
      do 405 i=1,nx-1
      do 406 j=1,ny2
      l=l+1
      v8(l,1)=v8(l,1)-v8(l+ny2,1)
      l2=l-1
      do 407 ii=i,nx-1
      jk=1
      if (ii.eq.i) jk=j
      do 407 jj=jk,ny2
      l2=l2+1
      v6(l,l2)=v7(l,l2)-v7(l+ny2,l2)-v7(l,l2+ny2)+v7(l+ny2,l2+ny2)
```

```
      v6(l2,l)=v6(l,l2)
  407 continue
  406 continue
  405 continue
      call st1(l,v8,v6,nb,v7,test(1,4))
c same thing for var-cov matrix:
      ny2=ny-1
      l=0
      do 411 i=1,nx
      lx=nfx+(i-1)*ny
      nyu=ny-1
      if (i.eq.nx) nyu=ny2-1
      do 412 j=1,nyu
      l=l+1
      l2=l-1
      do 413 i2=i,nx
      lx2=nfx+(i2-1)*ny
      nyu2=ny-1
      if (i2.eq.nx) nyu2=ny2-1
      jk=1
      if (i2.eq.i) jk=j
      do 413 j2=jk,nyu2
      l2=l2+1
      v8(l,l2)=vv(lx+j,lx2+j2)-vv(lx+j+1,lx2+j2)-vv(lx+j,lx2+j2+1)+
     &vv(lx+j+1,lx2+j2+1)
      v8(l2,l)=v8(l,l2)
  413 continue
      l2=l2+1
      v8(l,l2)=vv(lx+j,nfx+nb)-vv(lx+j+1,nfx+nb)
      v8(l2,l)=v8(l,l2)
  412 continue
  411 continue
      l=l+1
      v8(l,l)=vv(nfx+nb,nfx+nb)
c x-contrasts of y contrasts:
      l=0
      do 415 i=1,nx-1
      do 416 j=1,ny2
      l=l+1
      l2=l-1
      do 417 ii=i,nx-1
      jk=1
      if (ii.eq.i) jk=j
      do 417 jj=jk,ny2
      l2=l2+1
      v6(l,l2)=v8(l,l2)-v8(l+ny2,l2)-v8(l,l2+ny2)+v8(l+ny2,l2+ny2)
      v6(l2,l)=v6(l,l2)
  417 continue
  416 continue
  415 continue
      call st2(l,v6,v7,v8,nb,test(1,4),eig(1,4),iwork)
  100 format(5e15.8)
      return
      end

      subroutine st1(nn,beta,v6,nb,v7,test)
c calcs value of test statistic given beta and inv 2nd deriv matrix
      double precision v6(nb,nb),v7(nb,nb),test(2),beta(1)
      call pdi2(nn,v6,v7,nb,nb)
      test(1)=0
      do 27 ii=1,nn
      test(2)=0
      do 28 jj=1,ii
```

```
   28  test(2)=test(2)+v7(jj,ii)*beta(jj)
       test(1)=test(1)+test(2)*test(2)
   27  continue
       return
       end

       subroutine st2(nn,v6,v7,v8,nb,test,eig,iwork)
       double precision test(3),eig(1),v6(nb,nb),v7(nb,nb),v8(nb,nb)
       double precision qfg,trace(7)
       integer iwork(1)
c calculates eigenvalues and p-values, given chol decomp q-form (v7)
c and var-cov matrix (v6)
       do 18 i=1,nn
       do 18 j=i,nn
       v8(i,j)=0
       do 19 ii=1,i
       do 19 jj=1,j
   19  v8(i,j)=v8(i,j)+v7(ii,i)*v6(ii,jj)*v7(jj,j)
       v8(j,i)=v8(i,j)
   18  continue
       call eigen(v8,nn,nb,eig,v6,1,iflg)
       call sortg(eig,iwork,1,nn)
       do 50 ii=1,nn
       iwork(ii)=1
   50  v6(ii,1)=0
       test(2)=1-qfg(eig,v6,iwork,nn,0.d0,test(1),
      &5000,1.d-5,iwork(nn+1),trace,ifault)
       test(3)=0
       do 35 i=1,nb
   35  test(3)=test(3)+eig(i)
       return
       end

       subroutine cesti(nknx,nkny,tknot,np,beta,var,lvar,nex,ney,
      &est,nfx,ucx,ucy,scx,scy)
c designed to be used with output from sphi
c estimates are calculated at all combinations of equally spaced point
c s between xmin, xmax and ymin ymax.
c assumes cubic splines
c On output est contains the x and y ordinates (cols 1 and 2) and
c the lhr estimate (col 3) and its variance (col 4)
       double precision tknot(-2:nknx+3,2),beta(np),var(lvar,np)
       double precision est(nex*ney,4),a(4),b(4),xmin,xmax,ymin,ymax
       double precision dwork(16),ex,ucx,ucy,scx,scy
       integer iwork(16)
       l=0
       nky4=nkny+4
       nkx4=nknx+4
       xmin=tknot(0,1)
       ymin=tknot(0,2)
       xmax=tknot(nknx+1,1)
       ymax=tknot(nkny+1,2)
       do 10 i=1,nex
       ex=xmin+(i-1)*(xmax-xmin)/(nex-1)
       if (ex.gt.xmax) ex=xmax
       call spln(nknx,3,tknot(-2,1),ex,1,intx,a,1)
       do 15 j=1,ney
       l=l+1
       est(l,1)=ex
       est(l,2)=ymin+(j-1)*(ymax-ymin)/(ney-1)
       if (est(l,2).gt.ymax) est(l,2)=ymax
       call spln(nkny,3,tknot(-2,2),est(l,2),1,inty,b,1)
       ll=0
```

```
      do 21 ii=intx,intx+3
      if (ii.eq.nkx4.and.inty.eq.nkny+1) then
      l2=2
      else
      l2=3
      endif
      do 22 jj=inty,inty+l2
      ll=ll+1
      iwork(ll)=nfx+(ii-1)*nky4+jj
      dwork(ll)=a(ii-intx+1)*b(jj-inty+1)
   22 continue
   21 continue
      est(l,3)=0
      est(l,4)=0
      do 25 ii=1,ll
      est(l,3)=est(l,3)+beta(iwork(ii))*dwork(ii)
      do 25 jj=1,ll
      est(l,4)=est(l,4)+dwork(ii)*dwork(jj)*var(iwork(ii),iwork(jj))
   25 continue
      est(l,1)=scx*est(l,1)+ucx
      est(l,2)=scy*est(l,2)+ucy
   15 continue
   10 continue
      return
      end

      subroutine pencxc(nkx,tx,nky,ty,pen,wkx,wky)
calc pen matrix for 2 dim laplacian penalty
c note that this loops over y within x: ie parameters are
c ordered b(x1,y1),b(x1,y2),...,b(x1,y[nky+4]),b(x2,y1),...
c wky and wkx are workspace of length as given below
      double precision tx(-2:(nkx+4)),ty(-2:(nky+4)),
     &pen(nky+4,4,nkx+4,4),wkx(3,nkx+4,4),wky(3,nky+4,4),a1(4),a2(4),
     &a3(4),a4(4),tint(7)
      do 4 i=1,3
      do 4 j=1,nkx+4
      do 4 k=1,4
    4 wkx(i,j,k)=0
      do 5 i=1,3
      do 5 j=1,nky+4
      do 5 k=1,4
    5 wky(i,j,k)=0
      do 10 i=1,nkx+1
c i is looping over intervals,  The ith interval contributes to the
c (i,i),(i,i+1),(i,i+2),(i,i+3),(i+1,i+1),...,(i+3,i+3) pairs, since
c B(j) is >0 on I(j), I(j-1), I(j-2), I(j-3)
      call csplcf(nkx,tx,i,i,a1)
      call csplcf(nkx,tx,i+1,i,a2)
      call csplcf(nkx,tx,i+2,i,a3)
      call csplcf(nkx,tx,i+3,i,a4)
      call int1(tint,tx(i),tx(i-1))
      call int2(tint,a1,a1,wkx(1,i,1))
      call int2(tint,a1,a2,wkx(1,i,2))
      call int2(tint,a1,a3,wkx(1,i,3))
      call int2(tint,a1,a4,wkx(1,i,4))
      call int2(tint,a2,a2,wkx(1,i+1,1))
      call int2(tint,a2,a3,wkx(1,i+1,2))
      call int2(tint,a2,a4,wkx(1,i+1,3))
      call int2(tint,a3,a3,wkx(1,i+2,1))
      call int2(tint,a3,a4,wkx(1,i+2,2))
      call int2(tint,a4,a4,wkx(1,i+3,1))
   10 continue
c      write (65,100) (((wkx(i,j,k),i=1,3),j=1,nkx+4),k=1,4)
```

```
      do 20 i=1,nky+1
      call csplcf(nky,ty,i,i,a1)
      call csplcf(nky,ty,i+1,i,a2)
      call csplcf(nky,ty,i+2,i,a3)
      call csplcf(nky,ty,i+3,i,a4)
      call int1(tint,ty(i),ty(i-1))
      call int2(tint,a1,a1,wky(1,i,1))
      call int2(tint,a1,a2,wky(1,i,2))
      call int2(tint,a1,a3,wky(1,i,3))
      call int2(tint,a1,a4,wky(1,i,4))
      call int2(tint,a2,a2,wky(1,i+1,1))
      call int2(tint,a2,a3,wky(1,i+1,2))
      call int2(tint,a2,a4,wky(1,i+1,3))
      call int2(tint,a3,a3,wky(1,i+2,1))
      call int2(tint,a3,a4,wky(1,i+2,2))
      call int2(tint,a4,a4,wky(1,i+3,1))
   20 continue
c      write (66,100) (((wkx(i,j,k),i=1,3),j=1,nkx+4),k=1,4)
  100 format (e14.6)
      do 25 i=1,nky+4
      do 25 j=1,4
      do 25 k=1,nkx+4
      do 25 l=1,4
   25 pen(i,j,k,l)=0
      do 30 ix=1,nkx+4
      ixu=min(ix+3,nkx+4)-ix+1
      do 32 ixj=1,ixu
      do 35 iy=1,nky+4
      iyu=min(iy+3,nky+4)-iy+1
      do 37 iyj=1,iyu
      pen(iy,iyj,ix,ixj)=wkx(1,ix,ixj)*wky(3,iy,iyj)+
     &2*wkx(2,ix,ixj)*wky(2,iy,iyj)+wkx(3,ix,ixj)*wky(1,iy,iyj)
   37 continue
   35 continue
   32 continue
   30 continue
      return
      end

      subroutine int2(tint,a1,a2,b)
      double precision tint(7),a1(4),a2(4),b(3)
c  a1 are the coeficients of B1, a2 the coeficients of B2, on the
current interval.  tint is the output from int1
c  first calculate the contribution to B1*B2 term:
      b(1)=b(1)+a1(4)*a2(4)*tint(7)+(a1(3)*a2(4)+a1(4)*a2(3))*tint(6)+
     &(a1(2)*a2(4)+a1(3)*a2(3)+a1(4)*a2(2))*tint(5)+
     &(a1(1)*a2(4)+a1(2)*a2(3)+a1(3)*a2(2)+a1(4)*a2(1))*tint(4)+
     &(a1(1)*a2(3)+a1(2)*a2(2)+a1(3)*a2(1))*tint(3)+
     &(a1(1)*a2(2)+a1(2)*a2(1))*tint(2)+a1(1)*a2(1)*tint(1)
c  contribution to B1'*B2' term:
      b(2)=b(2)+a1(4)*a2(4)*tint(5)*9+(6*a1(3)*a2(4)+6*a1(4)*a2(3))*
     &tint(4)+(3*a1(2)*a2(4)+4*a1(3)*a2(3)+3*a1(4)*a2(2))*tint(3)+
     &(2*a1(2)*a2(3)+2*a1(3)*a2(2))*tint(2)+a1(2)*a2(2)*tint(1)
c  contribution to B1''*B2'' term:
      b(3)=b(3)+a1(4)*a2(4)*tint(3)*36+(12*a1(3)*a2(4)+12*a1(4)*a2(3))*
     &tint(2)+4*a1(3)*a2(3)*tint(1)
      return
      end

      subroutine int1(tint,txu,txl)
c txu is the knot at the upper limit of the interval
c txl the knot at the lower limit.  Computes the integral of t^j, for
c j=0,...,6 (stored in tint)
```

```
      double precision tint(7),txu,txl
      tint(1)=txu-txl
      tint(2)=tint(1)*(txu+txl)
      tint(3)=tint(2)*(txu+txl)-txu*txl*tint(1)
      tint(4)=tint(3)*(txu+txl)-txu*txl*tint(2)
      tint(5)=tint(4)*(txu+txl)-txu*txl*tint(3)
      tint(6)=tint(5)*(txu+txl)-txu*txl*tint(4)
      tint(7)=tint(6)*(txu+txl)-txu*txl*tint(5)
      do 20 j=2,7
   20 tint(j)=tint(j)/j
      return
      end

      subroutine penh2(nbx,nby,nbt,penm,mrx,mry,mdx,mdy,h,nmin,
     &ih,alpha)
c penm is the penalty matrix,  nbx and nby are the actual
c # x & y basis functions used, nbt is the total number used.
c in the configuration this is being written for, nbx would be #
c x-knots + 4, (sim for nby), but nbt would be nbx*nby-1 to avoid
c colinearity with the constant term (ie written for cubics with
c everything but the overall constant included in the tensor
c product term)
c h the neg sec deriv matrix
c nmin the entry in h where spline params begin.  alpha*P
c is added to the submatrix bounded by h(nm,nm)and h(nm+nbt-1,
c nm+nbt-1)
c mrx and mry are the actual row dimension of x and y parts of penm,
c and mdx and mdy are the # nonzero diags in each poriton
      double precision penm(mry,mdy,mrx,mdx),h(ih,ih),alpha
      nmax=nmin+nbt-1
      do 10 i=1,nbx
      l1=nmin+(i-1)*nby
      call penhg(nby,penm(1,1,i,1),mry,mdy,h,l1,l1,nmax,ih,alpha)
      l=1
      l2=l1
      m2=min(nbx,i+mdx-1)
      do 12 j=i+1,m2
      l=l+1
      l2=l2+nby
   12 call penhg(nby,penm(1,1,i,l),mry,mdy,h,l1,l2,nmax,ih,alpha)
   10 continue
      do 16 i=nmin,nmax-1
      do 16 j=i+1,nmax
   16 h(j,i)=h(i,j)
      return
      end

      subroutine penlk2(nbx,nby,nbt,beta,penm,mrx,mdx,mry,mdy,
     &lik,s,alpha)
c given a penalty matrix penm and parameter values beta,
c subtracts penalty terms from liklihood and score
c the beginning index of beta and s (the score) in the
c call should correspond to where the spline terms begin
c ie call penlik(...beta(7)...s(7)...) if the spline is
c stored consecutively beginning with the 7th component
c nk is the number of knots, nb is the number
c of basis fcns actually in use, alpha is the value of the
c smoothing parameter.
      double precision penm(mrx,mdx,mry,mdy),beta(1),s(1),lik,alpha
      nb2=nby-(nbx*nby-nbt)
      l1=1
      do 10 i=1,nbx
      m1=max(1,i-mdx+1)
```

```
      m2=min(nbx,i+mdx-1)
c      nb=nby
      l=0
      l1=1+(i-1)*nby
      l2=l1
      do 12 j=i,m2
      l=l+1
c      if (j.eq.nbx) nb=nb2
c l1 is the beginning row index, l2 the beginning column index
      nbr=nby
      nbc=nby
      if (i.eq.nbx) nbr=nb2
      if (j.eq.nbx) nbc=nb2
      call penlkg(nbr,nbc,beta(l2),beta(l1),penm(1,1,i,l),mry,mdy,lik,
     &s(l1),alpha)
      l2=l2+nby
  12  continue
      if (m1.lt.i) then
        l=i-m1+2
        do 11 j=m1,i-1
      l2=1+(j-1)*nby
        l=l-1
      nbr=nby
      nbc=nby
      if (i.eq.nbx) nbr=nb2
      if (j.eq.nbx) nbc=nb2
        call penlkg(nbr,nbc,beta(l2),beta(l1),penm(1,1,j,l),mry,mdy,lik,
     &s(l1),alpha)
  11    continue
      endif
  10  continue
      return
      end

      subroutine penhg(nb,penm,mr,md,h,nm1,nm2,max,ih,alpha)
c penm is the penalty matrix,  nb the actual
c # basis functions used, h the neg sec deriv matrix
c nmin the entry in h where spline params begin.  alpha*P
c is added to the submatrix bounded by h(nm1,nm2)and h(nm1+nb-1,
c nm2+nb-1)
c mr is the actual row dimension of penm, md is the # nonzero
c diagonals in the penalty matrix
c penm must be symmetric stored in the appropriate banded format
      double precision penm(mr,md),h(ih,ih),alpha,u
      do 10 i=1,nb
      i1=nm1+i-1
      i2=nm2+i-1
      if (i1.le.max.and.i2.le.max) then
      h(i1,i2)=h(i1,i2)+alpha*penm(i,1)
      endif
      if (i.lt.nb) then
        m2=min(nb,i+md-1)
        l=1
        do 12 j=i+1,m2
        j1=nm2+j-1
      j2=nm1+j-1
        l=l+1
        u=alpha*penm(i,l)
        if (i1.le.max.and.j1.le.max) then
          h(i1,j1)=h(i1,j1)+u
      endif
        if (j2.le.max.and.i2.le.max) then
        h(j2,i2)=h(j2,i2)+u
```

```
      endif
   12   continue
      endif
   10 continue
      return
      end

      subroutine penlkg(nb1,nb2,beta,beta2,penm,mr,md,lik,s,alpha)
c given a penalty matrix penm and parameter values beta,
c subtracts penalty terms from liklihood and score
c the beginning index of beta and s (the score) in the
c call should correspond to where the spline terms begin
c ie call penlik(...beta(7)...s(7)...) if the spline is
c stored consecutively beginning with the 7th component
c nk is the number of knots, nb1 is the number or rows of penm
c nb2 the # cols (in unbanded form) acutally used
c alpha is the value of the smoothing parameter.
c beta2 added to allow off center contributions.  That is, the beta
c parameters are used for the score and the complicated part of the
c liklihood, but multiplied by beta2 to get the likelihood contribution
c ie like contri = beta2' penm beta
      double precision penm(mr,md),beta(1),s(1),lik,alpha,u,beta2(1)
      do 10 i=1,nb1
      u=0
      m1=max(1,i-md+1)
      m2=min(nb2,i+md-1)
      l=0
      do 12 j=i,m2
      l=l+1
      u=u+penm(i,l)*beta(j)
   12 continue
      if (m1.lt.i) then
        l=i-m1+2
        do 11 j=m1,i-1
        l=l-1
        u=u+beta(j)*penm(j,l)
   11   continue
      endif
      s(i)=s(i)-alpha*u
      lik=lik-alpha*u*beta2(i)/2
   10 continue
      return
      end

      subroutine csplcf(nknot,t,j,k,a)
      double precision t(-2:(nknot+3)),a(0:3),a1,b1,c1,a2,b2,c2,t1,t2
c  t is the augmented knot sequence, nknot the number of knots
c  on interval I(k) the cubic spline basis function B(j) can be
c  represented as B(j)(t)=a(3)t^3+a(2)t^2+a(1)t+a(0)
c  this routine calculates a(3),a(2),a(1),a(0)
c      call dblepr("knots",5,t(-2),nknot+6)
      if (k.gt.j.or.k.lt.j-3) then
      a(0)=0
      a(1)=0
      a(2)=0
      a(3)=0
      else if (k.eq.j-3) then
      b1=1/(t(j-3)-t(j-4))
      a1=-t(j-4)*b1
      b2=1/(t(j-2)-t(j-4))
      a2=-t(j-4)*b2
      c1=b1*b2
      b1=b1*a2+b2*a1
```

```
      a1=a1*a2
      b2=1/(t(j-1)-t(j-4))
      a2=-t(j-4)*b2
      a(3)=c1*b2
      a(2)=c1*a2+b1*b2
      a(1)=b1*a2+a1*b2
      a(0)=a1*a2
      else if (k.eq.j-2) then
      b1=-1/(t(j-2)-t(j-3))
      a1=-t(j-2)*b1
      b2=-b1
      a2=t(j-3)*b1
      t1=1/(t(j-2)-t(j-4))
      t2=-1/(t(j-1)-t(j-3))
      c1=b1*t1+b2*t2
      b1=-b1*t(j-4)*t1+t1*a1-b2*t(j-1)*t2+t2*a2
c        b1=-(t(j-4)+t(j-2))*b1*t1-(t(j-1)+t(j-3))*t2*b2
      a1=-t(j-4)*t1*a1-t(j-1)*t2*a2
      c2=-t2*b2
      b2=-t2*a2+b2*t2*t(j-3)
      a2=a2*t2*t(j-3)
      t1=1/(t(j-1)-t(j-4))
      t2=-1/(t(j)-t(j-3))
      a(3)=t1*c1+t2*c2
      a(2)=t1*b1-t1*c1*t(j-4)+t2*b2-t2*c2*t(j)
      a(1)=t1*a1-t1*b1*t(j-4)+t2*a2-t2*b2*t(j)
      a(0)=-a1*t1*t(j-4)-t2*t(j)*a2
      else if (k.eq.j-1) then
      b1=-1/(t(j-1)-t(j-2))
      a1=-t(j-1)*b1
      b2=-b1
      a2=-b2*t(j-2)
      t1=1/(t(j-1)-t(j-3))
      t2=-1/(t(j)-t(j-2))
      c2=t2*b2+t1*b1
      b2=-t(j)*b2*t2+a2*t2-t1*t(j-3)*b1+a1*t1
      a2=-t(j)*t2*a2-t(j-3)*t1*a1
      c1=-t1*b1
      b1=t1*t(j-1)*b1-t1*a1
      a1=t1*t(j-1)*a1
      t1=1/(t(j-1)-t(j-4))
      t2=-1/(t(j)-t(j-3))
      a(3)=t1*c1+t2*c2
      a(2)=t1*b1-t1*c1*t(j-4)+t2*b2-t2*c2*t(j)
      a(1)=t1*a1-t1*b1*t(j-4)+t2*a2-t2*b2*t(j)
      a(0)=-a1*t1*t(j-4)-t2*t(j)*a2
      else if (k.eq.j) then
      b1=-1/(t(j)-t(j-1))
      a1=-t(j)*b1
      t1=-1/(t(j)-t(j-2))
      c1=t1*b1
      b1=t1*a1-b1*t1*t(j)
      a1=-t1*t(j)*a1
      t1=-1/(t(j)-t(j-3))
      a(3)=t1*c1
      a(2)=-c1*t1*t(j)+t1*b1
      a(1)=-t1*t(j)*b1+t1*a1
      a(0)=-t1*t(j)*a1
      endif
      return
      end
      subroutine intb1(nk,tk,o)
c subroutine to calculate the integral of cubic spline
```

```
c basis functions
c nk is the number of knots
      double precision tk(-2:nk+3),o(nk+4),a(4)
      do 10 i=1,nk+4
      o(i)=0
      kb1=max(1,i-3)
      kb2=min(i,nk+1)
c note: ith basis function defined on intervals kb1 to kb2
      do 11 j=kb1,kb2
      call csplcf(nk,tk,i,j,a)
c note: a(4) is the coef of the cubic term ... a(1) the coef of the const
      do 12 k=1,4
  12  o(i)=o(i)+a(k)*(tk(j)**k-tk(j-1)**k)/k
  11  continue
  10  continue
      return
      end

      subroutine stvc(nov,s,c,x,tknot,times,beta,lik0,lik,sb,vv,
     &iwork,dsub,isub,testr,eig,v2i,eps,z,exz,s0,s1,ts1,w,
     &wp,ts1p,id,s1p,z1)
      double precision s(1),c(1),x(1),tknot(1),times(1),beta(1),lik0
      double precision lik(2),sb(1),vv(1),dsub(1),testr(1),eig(1)
      double precision uplik,v2i(1)
C*************************************************************
      double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C*************************************************************
C>>>>>
      double precision wp(1),ts1p(1),s1p(1)
      integer id(1)
C<<<<<
      integer nov(15),iwork(1),isub(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      common /up/ uplik
c
c nov parameters:
c   1=no, 2=nrx(=no), 3=ncx, 4=istr (col # of x for strata, -1 if none,
c   5=maxiter, 6=knot optiont (1 use knots provided, 0 program calculates),
c   7=smoothing option (<0 use input smoothing params, 0 calc smoothing
c   param only in first iteration, >0 recalc after each iteration)
c   8=nfx (nfx+nsx in sph, nfx is #col in linear.cov, nsx in spline.cov),
c   9=nsx, 10=analysis option (<=0 estimates only, 1 est & var, >1 est,
c   var & test), 12=order of spline, 13=nest, 14=nknot, 15=ntime (nstimes
c   the number of points where covariates can switch values, >no for all
c   failure times)
c
c dsub must have length at least <<see below>>
c iwork must have length at least 3*no+max(no,np)+1+2*nstr+2*ntud*nstr
c where nstr is the # strata & ntud is  either ntime+1, or 1 if
c ntime>no.  Note that nsx is in nov(9)
c isub must have length 8+nsx+nfx+(n or ntud)
c testr must have length nsx*6
c eig: length nk4*2*nsx, will have eigenvalues for tests (output)
c if smoothing parameters are input need to be in components
c    nsx*2+1 to nsx*3 of dsub
c nov(12) is the order of the spline: 3 for cubic with squared second
c derivative penaly, 2 for quad & 1st deriv penalty, & 0 for constant
c with 1st diff penalty
c nov(13) is the number of point estimates when ntime>no
c estimates stored in dsub, which must be at least nest*(2*nsx+1) or
c (2*ntime+2)*(2*nsx+1)  beginning index will be dsub(isub(5))
c isub(5) is either 3*nsx+iord1*no+1 or 3*nsx+iord1*(ntime+1)+1
```

```
c v2i will have the inverse 2nd deriv matrix
c
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of splines
c isub(5) reserved for the beginning index (in dsub) of penm
c isub(6) reserved for the beginning index (in dsub) of unpenalized inf
c isub(7) reserved for number of knots
c isub(8) needs to be row dim of matrix of splines
c isub(8+1) to isub(8+nfx) gives col #'s of fixed covs
c isub(8+1+nfx) to isub(8+nfx+nsx) gives col #'s of spline covs
c isub(8+1+nfx+nsx) will be the beginning index of inter.
c
c ntime is as in coxrg: #switch points, not # intervals
c
c dsub(1:nsx) are target df.
c dsub(nsx+1:2*nsx) will be attained df.
c dsub(2*nsx+1:3*nsx) will be attained df.
c
c    calls stvc1, calvar, testw, cestb, cest directly
c  indirect calls: coxrg, coxft, coxg, strat, ut, uft, tint, tint4
c   cholg, solve sortg dperm sub5, sub5c, sub6c, eval5, act5, eval6, act6,
c   cks, gspl1, gspl2, pencb, penql, penco, extrct,
c   matop5, adisw, spln, penh, gdf, degf2, penlik, eigen,
c   qfg, pdi, pdi2
c
      no=nov(1)
      nrx=nov(2)
      ncx=nov(3)
      istr=nov(4)
      nknot=nov(14)
      ntime=nov(15)
      mit=nov(5)
      knopt=nov(6)
      ksmopt=nov(7)
      nfx=nov(8)
      nsx=nov(9)
      iord=nov(12)
      nest=nov(13)
      malph=2*nsx+1
      if (iord.ne.0.and.iord.ne.2.and.iord.ne.3) iord=0
      iord1=iord+1
      itk=-iord+1
      ncp=iord1
      if (iord.eq.0) then
      ntime=nknot
      ncp=2
      endif
      nbas=nknot+iord1
      nk2=nknot+2
      nk3=nknot+3
      nk4=nknot+4
      np=nfx+nsx*nbas
      nkk=max(nbas,np-nbas)
      kmd=(np-nbas)*(np-nbas)
      kmd=max(kmd,nbas*nbas+2*nbas*nkk)
c if ntime>no & ksmopt>0 then length(dsub)=3*nsx (df,smp)
c  +iord1*no (B-spline terms) (isub(4))
c  +ncp*nk4 (pen matrix) (isub(5))
c  +np*np   (unpen inf)  (isub(6))
c  +max((np-nbas)*(np-nbas),(nbas*nbas)+2*(nbas*max(nbas,np-nbas)))
c  +np*np+no+5*np      (work)
c if ntime>no and ksmopt<=0 then length(dsub)=3*nsx (df,smp)
c  +iord1*no (B-spline terms) (isub(4))
```

```
c  +ncp*nk4 (pen matrix) (isub(5))
c  +np*np   (unpen inf)  (isub(6))
c  +nbas*nbas+
c  +max(no,np*np+4*np)   (work)
c if ntime<no & ksmopt>0 then length(dsub)=3*nsx (df,smp)
c  +iord1*(ntime+1) (B-spline terms) (isub(4))
c  +ncp*nk4 (pen matrix) (isub(5))
c  +np*np   (unpen inf)  (isub(6))
c  +max((np-nbas)*(np-nbas),(nbas*nbas)+2*(nbas*max(nbas,np-nbas)))
c  +max(no,np*np+4*np)   (work)
c if ntime<no and ksmopt<=0 then length(dsub)=3*nsx (df,smp)
c  +iord1*(ntime+1) (B-spline terms) (isub(4))
c  +ncp*nk4 (pen matrix) (isub(5))
c  +np*np   (unpen inf)  (isub(6))
c  +nbas*nbas+
c  +max(no,np*np+4*np)      (work)
c  also, to accomodate est, must be at least 3*nsx+iord1*no+
c  nest*(2*nsx+1)  (ntime>no) or
c  (2*ntime+2)*(2*nsx+1)  (ntime<no)
      if (ntime.gt.no) then
       if (ksmopt.gt.0) then
           mdw=3*nsx+iord1*no+np*np+nk4*ncp+kmd+1
       else
           mdw=3*nsx+iord1*no+np*np+nk4*ncp+nbas*nbas+1
       endif
      else
       if (ksmopt.gt.0) then
           mdw=3*nsx+iord1*(ntime+1)+np*np+nk4*ncp+kmd+1
       else
           mdw=3*nsx+iord1*(ntime+1)+np*np+nk4*ncp+nbas*nbas+1
       endif
      endif
      call stvc1 (s,c,x,istr,isub,tknot,ntime,times,
     &beta,lik0,lik(1),sb,vv,iwork,dsub(mdw),dsub(1),eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      if (icnv.gt.0) then
         vv(1)=-icnv
         return
      endif
      lik(2)=uplik
      nov(15)=ntime
      nov(14)=nknot
c after call to calvar,
c done with dsub from isub(5) on out:
      mdw2=isub(5)
      mdw3=mdw2+nbas*nbas
      mdw4=mdw3+nbas*nbas
c (note:index inform in iwork partially overwritten in testw call)
      if (nov(10).gt.0) then
      call calvar(vv,np,np,dsub(isub(6)),np,vv,np,v2i)
      endif
      if (nov(10).gt.1) then
      call testw(np,nbas,nfx,nsx,beta,vv,np,v2i,np,
     &dsub(mdw2),dsub(mdw3),testr,iwork,dsub(mdw4),eig)
      endif
      if (ntime.gt.no) then
      call cest(nknot,tknot,np,beta,vv,np,tknot(3),tknot(nknot+4),
     &nest,dsub(isub(5)),nfx,nsx,iord)
      else
      call cestb(ntime,times,tknot(3),tknot(nknot+4),isub(isub(3)),
     &dsub(isub(4)),dsub(isub(5)),nfx,nsx,beta,vv,np,np,nknot,iord)
      endif
      return
```

```
      end
c
c for fitting time varying coefficient models
c file contains stvc1 gspl2 gspl1 eval5 eval6 evalc act5 act6 sub6c
c sub5 sub5c
c
c in addition, routines in this file call sortg, dperm, tint4, cks,
c   strat, tint, pencb, penql, penco, coxrg, spln, adisw, extrct, cholg,
c   matop5, gdf, penh, penlik
      subroutine stvc1(s,c,x,istr,isub,tknot,ntime,
     &times,beta,lik0,lik,sb,vv,iwork,work,dsub,eps,icnv,
     &z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      double precision vv(1),sb(1),work(1),beta(1),times(1),s(no)
      double precision dsub(1)
      double precision c(no),x(nrx,ncx),eps,lik,lik0,tknot(-2:nk3)
C*************************************************************
      double precision z(1),exz(1),s0(1),s1(1),ts1(1),w(1),z1(1)
C*************************************************************
C>>>>>
      double precision wp(1),ts1p(1),s1p(1)
      integer id(1)
C<<<<<
      integer iwork(1),isub(1)
      external eval5,eval6,act5,act6,sub5
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
c isub(3) reserved for the beginning index (in isub) of inter
c isub(4) reserved for the beginning index (in dsub) of splines
c isub(5) reserved for the beginning index (in dsub) of penm
c isub(6) reserved for the beginning index (in dsub) of unpenalized inf
c isub(7) reserved for number of knots
c isub(8) needs to be row dim of matrix of splines
c isub(8+1) to isub(8+nfx) gives col #'s of fixed covs
c isub(8+1+nfx) to isub(8+nfx+nsx) gives col #'s of spline covs
c isub(8+1+nfx+nsx) will be the beginning index of inter.
c
c ntime is as in coxrg: #switch points, not # intervals
c
c dsub(1:nsx) are target df.
c dsub(nsx+1:2*nsx) will be attained df.
c
c  first get knot locations:
c
      if (knopt.le.0.or.ntime.lt.no) then
        do 5 i=1,no
   5    iwork(i)=i
        call sortg(s,iwork,1,no)
        call dperm(c,iwork,1,no,work)
        do 6 j=1,ncx
   6    call dperm(x(1,j),iwork,1,no,work)
      endif
      knkp=nknot
      if (knopt.le.0) then
        ntud=nknot+1
        call tint4(ntud,s,c,tknot(1),no,iwork)
      if (iord.eq.0) then
        ntime=ntud-1
        do 717 kk=1,ntime
 717      times(kk)=tknot(kk)
      endif
        if (knkp.ne.ntud-1) then
          nknot=ntud-1
          nbas=nknot+iord1
```

```
          nk2=nknot+2
          nk3=nknot+3
          nk4=nknot+4
          np=nfx+nsx*nbas
        endif
        call cks(s(1),s(no),tknot(itk),nknot,iord)
      endif
c      write (6,*) (tknot(i),i=-2,(nknot+3))
c
c If ntime>no then spline functions will change at all failure times,
c otherwise, only at ntime times.
c
c this repeats some code from coxrg, but need to determine this first
c so splines can be calculated before calling coxrg.
c
      if (ntime.lt.no) then
      ntud=ntime+1
      if (times(ntime).le.0) then
        call tint4(ntud,s,c,times,no,iwork)
c note: if ntud gets altered in tint4 (because not enough failures
c then need to reset ntime to match.
        ntime=ntud-1
      endif
c     write (37,*) (times(i),i=1,ntime)
      else
      ntud=1
      endif
      mnstr=1
      miuti=mnstr+1
      mtduti=miuti+2*no
c miwk needs length max(no,np)
      miwk=mtduti+no
      mnostr=miwk+max(no,np)
      call strat(istr,s,c,x,iwork(mnostr),nstr,iwork(miwk),work,
     &no,ncx,nrx,iwork(miuti),iwork(mtduti))
c      write (6,*) nstr
      iwork(mnstr)=nstr
      mnuti=mnostr+nstr
      mitud=mnuti+nstr
c      write (6,*) (iwork(i),i=mnostr,mnostr+nstr-1)
c      write (6,*) (iwork(i),i=mnuti,mnuti+nstr-1)
c      write (6,*) ntime
      if (ntime.lt.no) then
         l1=1
         do 225 l1=1,nstr
         l2=(l1-1)*2*ntud
         call tint(ntud,times,s,c,iwork(mitud+12),no,iwork(miuti),
     &   iwork(mnuti+l1-1),l1)
         l1=iwork(mnostr+l1-1)+1
 225     continue
      endif
      isub(3)=9+nfx+nsx
      if (ksmopt.ge.0) then
      do 789 i=1,nsx
 789    dsub(i+2*nsx)=.1
      endif
      isub(4)=3*nsx+1
      if (ntime.gt.no) then
        isub(5)=isub(4)+iord1*no
        isub(8)=no
        call gspl1(tknot,s,nstr,iwork(miuti),iwork(mnuti),
     &  iwork(mnostr),isub(isub(3)),dsub(isub(4)))
      else
```

```
        isub(5)=isub(4)+iord1*ntud
        isub(8)=ntud
        call gspl2(tknot,s,nstr,iwork(miuti),iwork(mnuti),
     &  iwork(mtduti),iwork(mnostr),ntud,iwork(mitud),isub(isub(3)),
     &  dsub(isub(4)))
       endif
c isub(6) is the beginning index of the unpenalized inf matrix.
      isub(6)=ncp*nk4+isub(5)
      if (iord.eq.3) then
        call pencb(nknot,tknot(itk),dsub(isub(5)),nk4)
      else if (iord.eq.2) then
        call penql(nknot,tknot(itk),dsub(isub(5)),nk4,iord)
      else
        call penco(nknot,tknot(itk),dsub(isub(5)),nk4)
      endif
c      write (6,*) np
c      write (6,*) (isub(i),i=1,isub(3))
      do 28 i=1,np
   28 beta(i)=0
      if (ntime.gt.no) then
        call coxrg(s,c,x,no,nrx,ncx,istr,np,mit,1,ntime,times,beta,
     &  lik0,lik,sb,vv,iwork,work,act5,eval5,sub5,isub,dsub,eps,icnv,
     &  z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      else
        call coxrg(s,c,x,no,nrx,ncx,istr,np,mit,1,ntime,times,beta,
     &  lik0,lik,sb,vv,iwork,work,act6,eval6,sub5,isub,dsub,eps,icnv,
     &  z,exz,s0,s1,ts1,w,wp,ts1p,id,s1p,z1)
      endif
      return
      end

      subroutine gspl2(tk,s,nstr,iuti,nuti,tduti,nostr,ntud,
     &itud,inter,tt)
      double precision s(no),tk(-2:nk3),tt(ntud,iord1),b(4),smin,smax
      integer iuti(no,2),nostr(nstr),nuti(nstr),inter(ntud),tduti(no)
      integer itud(ntud,2,nstr)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      do 10 i=1,ntud
c  find midrange of failures in time interval
      iflg=0
      do 15 j=1,nstr
      l1=itud(i,1,j)
      l2=itud(i,2,j)
      if (l1.gt.0) then
        if (iflg.eq.0) then
          iflg=1
          smin=s(iuti(l1,1))
          smax=s(iuti(l2,1))
        else
          smin=min(smin,s(iuti(l1,1)))
          smax=max(smax,s(iuti(l2,1)))
        endif
      endif
   15 continue
      smin=(smin+smax)/2
      call spln(nknot,iord,tk(itk),smin,1,inter(i),b,1)
      do 725 k=1,iord1
  725 tt(i,k)=b(k)
   10 continue
      return
      end
```

```
      subroutine gspl1(tk,s,nstr,iuti,nuti,nostr,inter,tt)
      double precision s(no),tk(-2:(nk3)),tt(no,iord1),b(4)
      integer iuti(no,2),nostr(nstr),nuti(nstr),inter(no)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      nost=1
      do 724 ll=1,nstr
      do 723 j=nost,nuti(ll)
      l2=iuti(j,1)
      call spln(nknot,iord,tk(itk),s(l2),1,inter(l2),b,1)
      do 725 k=1,iord1
  725 tt(l2,k)=b(k)
  723 continue
      nost=nostr(ll)
  724 continue
      return
      end

      subroutine eval6(g,gp,b,lci,i,l,s,c,xx,nrx2,ncx2,no2,np2,lc2,nact,
     &iact,isub,dsub)
      double precision gp(np),s(no),c(no),xx(nrx,ncx),dsub(1),b(np),g
      integer isub(1),iact(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      call evalc(g,gp,b,isub(9),isub(9+nfx),nact,iact,
     &i,lci,xx,dsub(isub(4)),isub(8))
      return
      end

      subroutine eval5(g,gp,b,lci,i,l,s,c,xx,nrx2,ncx2,no2,np2,lc2,nact,
     &iact,isub,dsub)
      double precision gp(np),s(no),c(no),xx(nrx,ncx),dsub(1),b(np),g
      integer isub(1),iact(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      call evalc(g,gp,b,isub(9),isub(9+nfx),nact,iact,
     &i,l,xx,dsub(isub(4)),isub(8))
      return
      end

      subroutine evalc(g,gp,b,ifx,isx,nact,iact,i,l,x,tt,ntt)
c in this routine, i is the row of x to use, and l the row of tt. ntt
c is the row dim of tt
      double precision gp(1),tt(ntt,4),x(nrx,ncx),b(np),g
      integer iact(1),isx(nsx),ifx(nfx)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      do 20 ii=1,np
   20 gp(ii)=0
      g=0
      do 12 j=1,nfx
        gp(j)=x(i,ifx(j))
        g=g+gp(j)*b(j)
   12 continue
      do 13 j=1,nsx
        isp=iact(nfx+(j-1)*iord1+1)-1
        do 14 jj=1,iord1
          gp(isp+jj)=x(i,isx(j))*tt(l,jj)
          g=g+gp(isp+jj)*b(isp+jj)
   14   continue
   13 continue
  100 format(6e12.4)
      return
```

```
      end

      subroutine act6(lci,l1,l2,iuti,no2,s,c,x,nrx2,ncx2,iact,nact,
     &np2,lc2,isub,dsub)
      double precision s(no),c(no),x(nrx,ncx),dsub(1)
      integer iuti(no,2),iact(1),isub(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      isp=isub(isub(3)+lci-1)
      do 10 i=1,nfx
   10 iact(i)=i
      nact=nfx
      do 11 j=1,nsx
      isp2=nfx+nbas*(j-1)+isp
      do 12 jj=isp2,isp2+iord
      nact=nact+1
   12 iact(nact)=jj
   11 continue
      return
      end

      subroutine act5(lci,l1,l2,iuti,no2,s,c,x,nrx2,ncx2,iact,nact,
     &np2,lc2,isub,dsub)
      double precision s(no),c(no),x(nrx,ncx),dsub(1)
      integer iuti(no,2),iact(1),isub(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      isp=isub(isub(3)+iuti(l1,1)-1)
      do 10 i=1,nfx
   10 iact(i)=i
      nact=nfx
      do 11 j=1,nsx
      isp2=nfx+nbas*(j-1)+isp
      do 12 jj=isp2,isp2+iord
      nact=nact+1
   12 iact(nact)=jj
   11 continue
      return
      end

      subroutine sub6c(beta,lik,sb,vv,penm,v3,dft,dfa,
     &alpha,v6,v7,nit)
      double precision beta(np),lik,sb(np),vv(np,np),v3(np,np)
      double precision penm(nk4,ncp),dft(nsx),dfa(nsx),alpha(nsx)
      double precision v6(1),mindf,v7(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
c v6 and v7 must have length at least nk4*nk4
c first update values of smoothing parameters to give required df.
      if ((nit.eq.1.and.ksmopt.eq.0).or.(ksmopt.gt.0.and.nit.lt.9.and.
     &nit.gt.0)) then
      do 819 j=1,nsx
      l1=nfx+1+(j-1)*nbas
      l2=l1+nbas-1
      call adisw(l1,l2,np,vv,v3,np)
      dfa(j)=dft(j)
c  3rd from last arg in gdf # dcols used
c in pen matrix, 4th from last mindf (2 for cubic spline with 2nd deriv
c penalty)
      mindf=2.d0
      if (iord.lt.3) mindf=1.d0
      call gdf(nbas,penm,v3,v3,v6,np,dfa(j),alpha(j),v7,mindf,
     &ncp,nk4,iopp1)
```

```
c
c   if algorithm failed, return without adding penalty
c
      if (iopp1.gt.0) return
c      write (6,100) dft(j),dfa(j),alpha(j)
 819  continue
      endif
c then copy vv to v3:
      do 10 i=1,np
      do 10 j=1,np
      v3(i,j)=vv(i,j)
  10  continue
c then add penalty terms to lik,score, and inf (lik,sb,vv)
      do 535 j=1,nsx
      l1=nfx+1+(j-1)*nbas
      call penlik(nbas,beta(l1),penm,nk4,ncp,lik,sb(l1),alpha(j))
      call penh(nbas,penm,nk4,ncp,vv,l1,np,alpha(j))
 535  continue
 100  format (3e15.8)
      return
      end

      subroutine sub5(np2,beta,lik,sb,vv,vvi,isub,dsub,nit)
      double precision beta(np),lik,sb(np),vv(np,np),dsub(1)
      double precision vvi(np,np),uplik
      integer isub(1)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
      common /up/ uplik
c to provide working matrices, dsub must be at least
c 3*nsx+4*(n or ntud,depending on ntime)+np*np+2*(nknot+4)**2
      mv6=isub(6)+np*np
c      write (6,*) lik
      uplik=lik
c      call dblepr("lik",3,lik,1)
       if (ksmopt.gt.0.and.nsx.gt.1.and.nit.lt.9.and.nit.gt.0) then
      call sub5c(beta,lik,sb,vv,dsub(isub(5)),dsub(isub(6)),
     &dsub(1),dsub(1+nsx),dsub(malph),dsub(mv6),vvi)
        else
      call sub6c(beta,lik,sb,vv,dsub(isub(5)),dsub(isub(6)),
     &dsub(1),dsub(1+nsx),dsub(malph),dsub(mv6),vvi,nit)
      endif
c      write (6,*) lik
c      call dblepr("plik",4,lik,1)
 110  format (e20.12)
      return
      end

      subroutine sub5c(beta,lik,sb,vv,penm,v3,dft,dfa,alpha,v6,vvi)
      double precision beta(np),lik,sb(np),vv(np,np),v3(np,np)
      double precision penm(nk4,ncp),dft(nsx),dfa(nsx),alpha(nsx)
      double precision v6(1),mindf,vvi(np,np)
      common /paramt/ no,nrx,ncx,nknot,mit,knopt,ksmopt,np,nfx,nsx,
     &nk2,nk3,nk4,malph,iord,iord1,itk,ncp,nbas
c v6 must have length at least nbas*nbas+2*(max(nbas,np-nbas))*nbas
c first update values of smoothing parameters to give required df.
      mv8=1+nbas*nbas
      nkk=max(nbas,np-nbas)
      mv9=mv8+nbas*nkk
      do 820 i=1,np
      do 820 j=1,np
 820  vvi(i,j)=vv(i,j)
      do 821 j=2,nsx
```

```
      l1=nfx+1+(j-1)*nbas
      call penh(nbas,penm,nk4,ncp,vvi,l1,np,alpha(j))
 821  continue
      do 819 j=1,nsx
      l1=nfx+1+(j-1)*nbas
      l2=l1+nbas-1
      call extrct(vvi,np,np,l1-1,l2+1,v6,np-nbas)
      call cholg(np-nbas,v6,v3,np-nbas,np-nbas)
      call matop5(vv,np,v3,np-nbas,l1,l2,v6(mv8),nbas,v6(mv9),v6,nkk)
      dfa(j)=dft(j)
c  3rd from last arg in gdf # dcols used
c in pen matrix, 4th from last mindf (2 for cubic spline with 2nd deriv
c penalty)
      mindf=2.d0
      if (iord.lt.3) mindf=1.d0
      call gdf(nbas,penm,v6(mv9),v6(mv8),v6,nkk,dfa(j),alpha(j),
     &v3,mindf,ncp,nk4,iopp1)
c
c   if algorithm failed, return without adding penalty
c
      if (iopp1.gt.0) return
c      write (6,100) dft(j),dfa(j),alpha(j)
      call penh(nbas,penm,nk4,ncp,vvi,l1,np,alpha(j))
      if (j.lt.nsx) then
      call penh(nbas,penm,nk4,ncp,vvi,l1+nbas,np,-alpha(j+1))
      endif
 819  continue
c then copy vv to v3:
      do 10 i=1,np
      do 10 j=1,np
      v3(i,j)=vv(i,j)
  10  vv(i,j)=vvi(i,j)
c then add penalty terms to lik,score, and inf (lik,sb,vv)
      do 535 j=1,nsx
      l1=nfx+1+(j-1)*nbas
      call penlik(nbas,beta(l1),penm,nk4,ncp,lik,sb(l1),alpha(j))
c      call penh(nbas,penm,nk4,ncp,vv,l1,np,alpha(j))
 535  continue
 100  format (3e15.8)
      return
      end

      subroutine testw(np,nk4,nfx,nsx,beta,vv,lvv,v4,lv4,v6,v7,test,
     &iwork,v8,eig)
c For a spline with standard parametrization (all B-spline basis
c fcns included), tests (Wald) that all coefs=0 and that all coefs are
c equal. nk4 is the number of basis fcns per spline term. nfx the
c number of nonspline covs (come first in beta) and nsx the number
c of spline terms (parameters assumed to be consecutive in beta)
c so np=nfx+nsx*nk4
c on input v4 is the inverse penalized 2nd derivative matrix, and vv the
c estimated var-cov matrix of the paramters estimates.
c  length(iwork)=2*nk4
c v6,v7,v8 are work space,
c on output eig will contain the eigenvalues and test the test
c results
c calls pdi2, sortg, eigen qfg from ~/src/util.a
      double precision vv(lvv,np),beta(np),v4(lv4,np)
      double precision v6(nk4,nk4),v7(nk4,nk4),test(6,nsx)
      double precision v8(nk4,nk4),qfg,trace(7),eig(nk4,2*nsx)
      integer iwork(1)
      do 50 k=1,nsx
      l1=nfx+(k-1)*nk4
```

```
c  overall test for var i:
c  first calc stat:
      do 15 i=1,nk4
      do 15 j=1,nk4
      v6(i,j)=v4(i+l1,j+l1)
   15 continue
      call pdi2(nk4,v6,v7,nk4,nk4)
      test(1,k)=0
      do 27 ii=1,nk4
      test(2,k)=0
      do 28 jj=1,ii
   28 test(2,k)=test(2,k)+v7(jj,ii)*beta(l1+jj)
      test(1,k)=test(1,k)+test(2,k)*test(2,k)
   27 continue
c then get eigenvalues to calc dist
      do 17 i=1,nk4
      do 17 j=1,nk4
   17 v6(i,j)=vv(i+l1,j+l1)
      do 18 i=1,nk4
      do 18 j=i,nk4
      v8(i,j)=0
      do 19 ii=1,i
      do 19 jj=1,j
   19 v8(i,j)=v8(i,j)+v7(ii,i)*v6(ii,jj)*v7(jj,j)
      v8(j,i)=v8(i,j)
   18 continue
      call eigen(v8,nk4,nk4,eig(1,2*k-1),v6,1,iflg)
      call sortg(eig(1,2*k-1),iwork,1,nk4)
      do 250 ii=1,nk4
      iwork(ii)=1
  250 v6(ii,1)=0
      test(2,k)=1-qfg(eig(1,2*k-1),v6,iwork,nk4,0.d0,test(1,k),
     &5000,1.d-5,iwork(nk4+1),trace,ifault)
      test(3,k)=0
      do 35 i=1,nk4
   35 test(3,k)=test(3,k)+eig(i,2*k-1)
c  test for proportional hazards:
c  note that eigenvalues the same as the first nk4-1 eigs used
c  before. (actually not always)
      nk3=nk4-1
      do 215 i=1,nk4
      do 215 j=1,nk4
  215 v6(i,j)=v4(i+l1,j+l1)
      do 217 i=1,nk3
      do 217 j=1,nk4
  217 v6(i,j)=v6(i,j)-v6(i+1,j)
      do 218 j=1,nk3
      v8(j,1)=beta(l1+j)-beta(l1+j+1)
      do 218 i=1,nk3
  218 v6(i,j)=v6(i,j)-v6(i,j+1)
      call pdi2(nk3,v6,v7,nk4,nk4)
      test(4,k)=0
      do 227 ii=1,nk3
      test(5,k)=0
      do 228 jj=1,ii
  228 test(5,k)=test(5,k)+v7(jj,ii)*v8(jj,1)
      test(4,k)=test(4,k)+test(5,k)*test(5,k)
  227 continue
c get eigenvalues, since actually not the same if more than 1 covariate
c modelled with splines
      do 237 i=1,nk4
      do 237 j=1,nk4
  237 v6(i,j)=vv(i+l1,j+l1)
```

```
      do 235 i=1,nk3
      do 235 j=1,nk4
  235 v6(i,j)=v6(i,j)-v6(i+1,j)
      do 236 j=1,nk3
      do 236 i=1,nk3
  236 v6(i,j)=v6(i,j)-v6(i,j+1)
      do 238 i=1,nk3
      do 238 j=i,nk3
      v8(i,j)=0
      do 239 ii=1,i
      do 239 jj=1,j
  239 v8(i,j)=v8(i,j)+v7(ii,i)*v6(ii,jj)*v7(jj,j)
      v8(j,i)=v8(i,j)
  238 continue
      call eigen(v8,nk3,nk4,eig(1,2*k),v6,1,iflg)
      call sortg(eig(1,2*k),iwork,1,nk3)
      do 240 i=1,nk3
  240 v6(i,1)=0
      test(5,k)=1-qfg(eig(1,2*k),v6,iwork,nk3,0.d0,test(4,k),5000,
     &1.d-5,iwork(nk4+1),trace,ifault)
      test(6,k)=0
      do 335 i=1,nk4
  335 test(6,k)=test(6,k)+eig(i,2*k)
   50 continue
  100 format(5e15.8)
      return
      end

      subroutine cest(nknot,tknot,np,beta,var,lvar,smin,smax,nest,est,
     &nfx,nsx,iord)
c splines contain all B-spline terms
c estimates are calculated at nest equally spaced points from
c smin to smax
c c iord is the order of the spline (3 for cubic)
c On output est contains the time ordinates (col 1) and
c the lhr estimate (col 2*j) and its variance (col 2*j+1)
      double precision tknot(-2:nknot+4),beta(np),var(lvar,np)
      double precision est(nest,1+2*nsx),b(4),smin,smax
      ior=iord+1
      nbas=nknot+ior
      do 10 i=1,nest
      est(i,1)=smin+(i-1)*(smax-smin)/(nest-1)
      call spln(nknot,iord,tknot(1-iord),est(i,1),1,int,b,1)
      do 12 j=1,nsx
      l1=nfx+(j-1)*nbas+int-1
      j1=j*2
      j2=j1+1
      est(i,j1)=0
      est(i,j2)=0
      do 17 i1=1,ior
      est(i,j1)=est(i,j1)+b(i1)*beta(l1+i1)
      do 17 k1=1,ior
      est(i,j2)=est(i,j2)+b(i1)*b(k1)*var(l1+i1,l1+k1)
   17 continue
   12 continue
   10 continue
      return
      end

      subroutine cestb(ntime,times,smin,smax,inter,tt,est,nfx,nsx,
     &beta,var,lvar,np,nknot,iord)
c splines contain all B-spline terms (intervals in inter, B-splines in
c tt) in consecutive entries in beta and var (coefs and variances)
```

```
c iord is the order of the spline (3 for cubic)
c smin and smax are min and max values for plot, other step fcn
c boundries given by times. nfx is number fixed and nsx # spline
c terms.  On output est contains the time ordinates (col 1) and
c the lhr estimate (col 2*j) and its variance (col 2*j+1)
      double precision times(ntime),smin,smax,tt(ntime+1,4)
      double precision est(2*(ntime+1),1+2*nsx),beta(np),var(lvar,np)
      integer inter(1)
      ior=iord+1
      nbas=nknot+ior
      est(1,1)=smin
      do 5 j=1,nsx
      l1=nfx+(j-1)*nbas+inter(1)-1
      j1=2*j
      j2=2*j+1
      est(1,j1)=0
      est(1,j2)=0
      do 17 i1=1,ior
      est(1,j1)=est(1,j1)+tt(1,i1)*beta(l1+i1)
      do 17 k1=1,ior
      est(1,j2)=est(1,j2)+tt(1,i1)*tt(1,k1)*var(l1+i1,l1+k1)
 17   continue
  5   continue
      do 10 i=2,ntime+1
      ii1=2*(i-1)
      est(ii1,1)=times(i-1)
      do 11 j=1,nsx
      est(ii1,2*j)=est(ii1-1,2*j)
 11   est(ii1,2*j+1)=est(ii1-1,2*j+1)
      ii1=ii1+1
      est(ii1,1)=est(ii1-1,1)
      do 6 j=1,nsx
      l1=nfx+(j-1)*nbas+inter(i)-1
      j1=2*j
      j2=2*j+1
      est(ii1,j1)=0
      est(ii1,j2)=0
      do 18 i1=1,ior
      est(ii1,j1)=est(ii1,j1)+tt(i,i1)*beta(l1+i1)
      do 18 k1=1,ior
      est(ii1,j2)=est(ii1,j2)+tt(i,i1)*tt(i,k1)*var(l1+i1,l1+k1)
 18   continue
  6   continue
 10   continue
      ii1=2*(ntime+1)
      est(ii1,1)=smax
      do 21 j=1,nsx
      est(ii1,2*j)=est(ii1-1,2*j)
 21   est(ii1,2*j+1)=est(ii1-1,2*j+1)
      return
      end

      subroutine intsb2(ntud,s,t,no)
c  subroutine to determine knots . Data must be sorted on s
c ntud is the # of knots (input).
c   Ij is (t(j-1),t(j)].  t(0) is set to s(1) and t(ntud+1) to s(no)
      double precision s(no),t(0:(ntud+1))
      nft=no
      nfk=max(int(nft/float(ntud+1)+.5),1)
      t(0)=s(1)
      l1=1
      k=0
 15   if (l1.ge.no) then
```

```
      t(k+1)=s(no)
      if (t(k+1).eq.t(k)) then
       ntud=k-1
      else
       ntud=k
      endif
c     write (6,100) ntud
 100    format ('# nkots reset in intsub to',i5)
      return
      endif
      if (t(k).lt.s(l1+1)) then
        k=k+1
      nft=no-l1
        nfk=max(int(nft/float(ntud-k+2)+.5),1)
      l1=l1+nfk
      t(k)=s(l1)
        if (k.eq.ntud) then
        t(k+1)=s(no)
        if (t(k+1).eq.t(k)) ntud=k-1
          return
      else
        go to 15
      endif
      else
        l1=l1+1
        go to 15
      endif
      end

      subroutine spln(nk,m,wk,y,n,inter,x,lx)
c nk # knots, m=highest degree polynomial in spline(0 for
c const, 3 for cubic), wk(1) to wk(nk)=interior knots (ordered smallest
c to largest), wk(-m+1) to wk(nk+m) must be augmented knot sequence
c on output x will be an (nx[m+1]) matrix giving the values of the
c active basis functions at each point.  inter(ii) gives
c which interval y(ii) falls in (y is the vector of input points)
c   knot intervals are of the form Ij=(t(j-1),tj].
c n is the number of points where values are calculated
c lx is actual row dim of x.
c NOTE: if m=0 then only uses interior knots 1 to nk.  If
c  m>0 then knots bounding the range of y also need to be included.
c  if m>0 and y<wk(0) or y>wk(nk+1) then returns all 0's for inter
c  and x(ii,.)
      double precision y(n),x(lx,m+1),wk(-m+1:nk+m),bj0,bj1
      double precision hj0,hj1
      integer inter(n)
      nb=nk+m+1
      l=1
      do 99 ii=1,n
      if (m.gt.0.and.(y(ii).lt.wk(0).or.y(ii).gt.wk(nk+1))) then
      inter(ii)=0
      do 5 jj=1,m+1
   5    x(ii,jj)=0
      l=1
      go to 99
      endif
      do 10 j=1,m+1
  10  x(ii,j)=0
  22  if (y(ii).gt.wk(l)) then
        l=l+1
        go to 22
      endif
      inter(ii)=1
```

```
      x(ii,1)=1
      if (m.gt.0) then
        do 25 k=1,m
        max=l+k
        lc=1+k
        x(ii,lc)=x(ii,lc-1)*(y(ii)-wk(max-k-1))/(wk(max-1)-wk(max-k-1))
        if (k.gt.1) then
          do 26 k1=max-1,l+1,-1
          k3=k1-l+1
          bj1=x(ii,k3)
          hj1=(wk(k1)-y(ii))/(wk(k1)-wk(k1-k))
          bj0=x(ii,k3-1)
          hj0=(y(ii)-wk(k1-k-1))/(wk(k1-1)-wk(k1-k-1))
          x(ii,k3)=hj0*bj0+hj1*bj1
 26       continue
        endif
        x(ii,1)=x(ii,1)*(wk(l)-y(ii))/(wk(l)-wk(l-k))
 25     continue
      endif
      if (ii.lt.n) then
      if (y(ii+1).lt.y(ii)) l=1
      endif
 99   continue
100   format (6e13.6)
      return
      end

      subroutine cks(smin,smax,tknot,nknot,iosp)
c soubroutine to Complete the Knot Sequence.
c smin and smax are smallest and largest values of var modelled by spline
c tknot has interior knots in positions 1 to nknot
c iosp is the power of the largest polynomial in the spline
c (ie 3 for cubic)
      double precision smin,smax,tknot((1-iosp):(nknot+iosp))
      ntud=nknot+1
      if (smin.lt.tknot(1)) then
      tknot(0)=smin
      else
      tknot(0)=tknot(1)-1
      endif
      if (smax.gt.tknot(nknot)) then
      tknot(ntud)=smax
      else
      tknot(ntud)=tknot(nknot)+1
      endif
      do 717 jj=1,iosp-1
      tknot(ntud+jj)=tknot(ntud)
      tknot(-jj)=tknot(0)
717   continue
      return
      end

      subroutine penco(nk,wk,penm,mr)
c nk is # knots, wk is augmented knot seq.
c This subroutine calculates the penalty matrix for integrated squared
c first derivative penalty, for quadratic spline.
c  The first col of penm will have the main
c diagonal, the 2nd col the next diag ... the 3rd col the 2nd diag from
c the main.
c mr is the actual row dim of penm
c  iosp is the order of the spline.  must be 2 for this routine
      double precision wk(1:nk),penm(mr,2)
      nb=nk+1
```

```
      penm(1,1)=1
      penm(nb,2)=0
      penm(nb,1)=1
      penm(1,2)=-1
      do 20 i=2,nk
      penm(i,1)=2
      penm(i,2)=-1
   20 continue
      return
      end

      subroutine penql(nk,wk,penm,mr,iosp)
c nk is # knots, wk is augmented knot seq.
c This subroutine calculates the penalty matrix for integrated squared
c first derivative penalty, for quadratic spline.
c  The first col of penm will have the main
c diagonal, the 2nd col the next diag ... the 3rd col the 2nd diag from
c the main.
c mr is the actual row dim of penm
c  iosp is the order of the spline.  must be 2 for this routine
      double precision wk(-(iosp-1):nk+iosp),penm(mr,(iosp+1)),tl(4),
     &tu(4),a(4),b(4)
      double precision t1,t2,t3,t4
      nb=nk+iosp+1
      nw=iosp+1
      do 10 i=1,nb
      do 10 j=1,nw
   10 penm(i,j)=0
      tl(1)=-2/(wk(1)-wk(-1))
      tl(2)=-tl(1)
      tl(3)=0
      do 20 l=1,nk+1
c tl has the value of the 2nd deriv at the lower endpoint of the lth
c interval, tu at the upper (2nd derivs of basis fcns are linear on
c each interval
      t1=wk(l)-wk(l-1)
      t4=wk(l)+wk(l-1)
      t2=t1*t4/2
      t3=t1*(t4*t4-wk(l)*wk(l-1))/3
c note that t3 is (wk(l)^3-wk(l-1)^3)/3
      tu(1)=0
      tu(2)=-4/(wk(l+1)-wk(l-1))
      if (l.eq.nk+1) tu(2)=tu(2)/2
      tu(3)=-tu(2)
c calc slopes and intercepts on interval l:
      do 21 j=1,nw
      b(j)=(tu(j)-tl(j))/(wk(l)-wk(l-1))
      a(j)=tu(j)-b(j)*wk(l)
   21 continue
      do 22 j=1,nw
      ll=l+j-1
      j2=j-1
      do 22 k=j,nw
      penm(ll,k-j2)=penm(ll,k-j2)+2*(a(j)*a(k)*t1+(a(j)*b(k)+
     &a(k)*b(j))*t2+b(j)*b(k)*t3)
   22 continue
      tl(1)=tu(2)
      tl(2)=tu(3)
      tl(3)=0
   20 continue
      return
      end
```

```
      subroutine dwch(e,c,ndf,k,stat,pv,iwk)
c subroutine to call function qfg (in dwchis)
c k is the # terms
c e is the eigenvalues (weights in weighted sum)
c c the noncentrality parameters, ndf the number of
c degrees of freedom.  Note the sigma term in qfg is assumed to
c be 0 to simplify this call
c stat is the value of the quad form, calcs prob>stat and puts the
c result in pv.
c for interp of ifault and trace see qfg
      double precision e(k),c(k),stat,pv,trace(7),acc,qfg,sig
      integer ndf(k),iwk(k),ifault
      acc=1.d-5
      sig=0.d0
      lim=10000
      pv=1-qfg(e,c,ndf,k,sig,stat,lim,acc,iwk,trace,ifault)
      return
      end

      double precision FUNCTION QFg(ALB,ANC,N,IRR,SIGMA,CC,LIM1,
     &ACC,ITH,TRACE,IFAULT)
c*** from statlib, Wed Dec 12 08:23:41 EST 1990 ***
C
C     ALGORITHM AS 155  APPL. STATIST. (1980) VOL.29, NO.3
C
C     Distribution of a linear combination of non-central chi-squared
C     random variables.
C
c  calcs dist of sum(alb(j)*X(j))+sigma*X(0) where X(j) is chi-square
c with n(j) df and noncent par anc(j), and X(0) is std normal
c  (hopefully) converted to dble precision (no real reason to do this
c   since don't comput results to that accuracy anyway)
c  args:  alb input values of weights
c  anc input values of noncentrality parameters
c  n input integer values of df
c irr input # chi-square terms (lenght of alb,anc,n)
c sigma, see formula above
c cc point at which df is evaluated
c lim1 max # terms in integral
c acc requested accuracy
c ith is an integer working vector.
c output: trace(1) abs val sum trace (2) total # int terms (3) number of
c integrations (4) int interval i main integration (5) truncaiton point
c in main integration (6) std dev of conv factor (7) number cycles to
c locate integ params; ifault=0 ok, =1 req acc not obtained, =2 round
c off error may be sig (this is computed based on single precision so
c the dble precision calcs here are hopefully ok), =3 invalid params
c =4 unable to locate int params.
      implicit double precision (a-h,o-z)
      INTEGER IRR,LIM1,IFAULT
      double precision SIGMA,CC,ACC
      double precision TRACE(7),ALB(IRR),ANC(IRR)
      INTEGER N(IRR),ITH(IRR)
      INTEGER J,NJ,NT,NTM
      double precision ACC1,ALMX,XLIM,XNT,XNTM
      double precision UTX,TAUSQ,SD,AINTV,AINTV1,X,UP,UN,D1,D2,ALJ,ANCJ
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,HALF,
     &ONE,TWO,FOUR,
     1  SIXTN,FOURP5,PT07,PT2,QUART,TEN,PT33,PT67,PT75,ONEPT5,THREE,
     2  ONEPT1
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
```

```
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO /0.0d0/,HALF/0.5d0/,ONE/1.0d0/,TWO/2.0d0/,FOUR/4.0d0/,
     &SIXTN/16.0d0/,FOURP5/4.5d0/,PT07/0.07d0/,PT2/0.2d0/,
     &QUART/0.25d0/,TEN/10.0d0/,PT33/0.33d0/,PT67/0.67d0/,PT75/0.75d0/,
     &ONEPT5/1.5d0/,THREE/3.0d0/,ONEPT1/1.1d0/
C
      IROUND(X) = INT(X + SIGN(HALF,X))
C
C     Setting constants in COMMON.   ALN28 = ln(2) / 8.
C
      PI = 3.14159265358979
      ALN28 = 0.08664339758
C
      C = CC
      IR = IRR
      LIM = LIM1
      DO 10 J = 1,7
      TRACE(J) = ZERO
 10   CONTINUE
      IFAULT = 0
      ICOUNT = 0
      AINTL = ZERO
      ERSM = ZERO
      QFg = -ONE
      ACC1 = ACC
      NDTSRT = .TRUE.
      FAIL = .FALSE.
C
C     Find mean, sd, max & min of ALB.
C     Check that parameter values are valid.
C
      XLIM = LIM
      SIGSQ = SIGMA**2
      SD = SIGSQ
      ALMAX = ZERO
      ALMIN = ZERO
      AMEAN = ZERO
      J = 1
 20   IF (.NOT.(J.LE.IR)) GO TO 60
      NJ = N(J)
      ALJ = ALB(J)
      ANCJ = ANC(J)
      IF (.NOT.(NJ.LT.0.OR.ANCJ.LT.ZERO)) GO TO 30
      IFAULT = 3
      GO TO 260
 30   SD = SD + ALJ**2*(2*NJ + FOUR*ANCJ)
      AMEAN = AMEAN + ALJ*(NJ + ANCJ)
      IF (.NOT.(ALMAX.LT.ALJ)) GO TO 40
      ALMAX = ALJ
      GO TO 50
 40   IF (.NOT.(ALMIN.GT.ALJ)) GO TO 50
      ALMIN = ALJ
 50   J = J + 1
      GO TO 20
 60   IF (.NOT.(SD.EQ.ZERO)) GO TO 80
      IF (.NOT.(C.GT.ZERO)) GO TO 70
      QFg = ONE
      GO TO 260
 70   QFg = ZERO
      GO TO 260
 80   IF (.NOT.(ALMIN.EQ.ZERO.AND.ALMAX.EQ.ZERO.AND.SIGMA.EQ.ZERO))
     1  GO TO 90
```

```
      IFAULT = 3
      GO TO 260
 90   SD = SQRT(SD)
      IF (.NOT.(ALMAX.LT.-ALMIN)) GO TO 100
      ALMX = -ALMIN
      GO TO 110
 100  ALMX = ALMAX
C
C     Starting values for FINDU * CTFF.
C
 110  UTX = SIXTN/SD
      UP = FOURP5/SD
      UN = -UP
C
C     Truncation point with no convergence factor.
C
      CALL FINDU (N,ALB,ANC,UTX,HALF*ACC1)
C
C     Does convergence factor help ?
C
      IF (.NOT.(C.NE.ZERO.AND.ALMX.GT.PT07*SD)) GO TO 130
      TAUSQ = QUART*ACC1/CFE(N,ALB,ANC,ITH,C)
      IF (.NOT.(FAIL)) GO TO 120
      FAIL = .FALSE.
      GO TO 130
 120  IF (.NOT.(TRUNCN(N,ALB,ANC,UTX,TAUSQ).LT.PT2*ACC1)) GO TO 130
      SIGSQ = SIGSQ + TAUSQ
      CALL FINDU (N,ALB,ANC,UTX,QUART*ACC1)
      TRACE(6) = SQRT(TAUSQ)
 130  TRACE(5) = UTX
      ACC1 = HALF*ACC1
C
C     Find 'range' of distribution, quit if outside of this.
C
 140  D1 = CTFF(N,ALB,ANC,ACC1,UP) - C
      IF (.NOT.(D1.LT.ZERO)) GO TO 150
      QFg = ONE
      GO TO 260
 150  D2 = C - CTFF(N,ALB,ANC,ACC1,UN)
      IF (.NOT.(D2.LT.ZERO)) GO TO 160
      QFg = ZERO
      GO TO 260
C
C     Find integration interval.
C
 160  IF (.NOT.(D1.GT.D2)) GO TO 170
      AINTV = D1
      GO TO 180
 170  AINTV = D2
 180  AINTV = TWO*PI/AINTV
C
C     Calculate number of terms required for main & auxiliary
C     integrations.
C
      XNT = UTX/AINTV
      XNTM = THREE/SQRT(ACC1)
      IF (.NOT.(XNT.GT.XNTM*ONEPT5)) GO TO 220
      IF (.NOT.(XNTM.GT.XLIM)) GO TO 190
      IFAULT = 1
      GO TO 260
C
C     Parameters for auxiliary integration.
C
```

```
 190  NTM = IROUND(XNTM)
      AINTV1 = UTX/XNTM
      X = TWO*PI/AINTV1
      IF (.NOT.(X.LE.ABS(C))) GO TO 200
      GO TO 220
C
C     Calculate convergence factor.
C
 200  TAUSQ = CFE(N,ALB,ANC,ITH,C - X) + CFE(N,ALB,ANC,ITH,C + X)
      TAUSQ = PT33*ACC1/(ONEPT1*TAUSQ)
      IF (.NOT.(FAIL)) GO TO 210
      GO TO 220
 210  ACC1 = PT67*ACC1
C
C     Auxiliary integration.
C
      CALL INTEGR (N,ALB,ANC,NTM,AINTV1,TAUSQ,.FALSE.)
      XLIM = XLIM - XNTM
      SIGSQ = SIGSQ + TAUSQ
      TRACE(3) = TRACE(3) + 1
      TRACE(2) = TRACE(2) + NTM + 1
C
C     Find truncation point with new convergence factor.
C
      CALL FINDU (N,ALB,ANC,UTX,QUART*ACC1)
      ACC1 = PT75*ACC1
      GO TO 140
C
C     Main integration.
C
 220  TRACE(4) = AINTV
      IF (.NOT.(XNT.GT.XLIM)) GO TO 230
      IFAULT = 1
      GO TO 260
 230  NT = IROUND(XNT)
      CALL INTEGR (N,ALB,ANC,NT,AINTV,ZERO,.TRUE.)
      TRACE(3) = TRACE(3) + 1
      TRACE(2) = TRACE(2) + NT + 1
      QFg = HALF - AINTL
      TRACE(1) = ERSM
      UP = ERSM
C
C     Test whether round-off error could be significant.
C     Allow for radix 8 or 16 machines.
C
      X = UP + ACC/TEN
      J = 1
 240  IF (.NOT.(J.LE.8)) GO TO 260
      IF (.NOT.(J*X.EQ.J*UP)) GO TO 250
      IFAULT = 2
 250  J = J*2
      GO TO 240
 260  TRACE(7) = ICOUNT
      RETURN
      END
C
      SUBROUTINE GCNTR
C
C     Count number of calls to ERRBD, TRUNCN & CFE.
C
      implicit double precision (a-h,o-z)
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C
```

```
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
C
      ICOUNT = ICOUNT + 1
      IF (.NOT.(ICOUNT.GT.LIM)) GO TO 20
c      WRITE (6,10)
c 10   FORMAT (' qfg: cannot locate integration parameters'/)
c      STOP
      call intpr("emerg stop in qfg",17,icount,1)
      call intpr("maybe in inf loop",17,icount,1)
 20   RETURN
      END
C
      double precision FUNCTION ALOG1(X,FIRST)
C
C     If FIRST then return ln(1 + x) else ln(1 + x) - x.
C
      implicit double precision (a-h,o-z)
      double precision X
      LOGICAL FIRST
      double precision S,S1,TERM,Y,AK,PT1,ONE,TWO,THREE
      DATA PT1/0.1d0/,ONE/1.0d0/,TWO/2.0d0/,THREE/3.0d0/
C
      F1(I) = S + TERM/AK
C
      IF (.NOT.(ABS(X).GT.PT1)) GO TO 20
      IF (.NOT.(FIRST)) GO TO 10
      ALOG1 = LOG(ONE + X)
      GO TO 70
 10   ALOG1 = LOG(ONE + X) - X
      GO TO 70
 20   Y = X/(TWO + X)
      TERM = TWO*Y**3
      AK = THREE
      IF (.NOT.(FIRST)) GO TO 30
      S = TWO
      GO TO 40
 30   S = -X
 40   S = S*Y
      Y = Y**2
      S1 = F1(0)
 50   IF (.NOT.(S1.NE.S)) GO TO 60
      AK = AK + TWO
      TERM = TERM*Y
      S = S1
      S1 = F1(0)
      GO TO 50
 60   ALOG1 = S
 70   RETURN
      END
C
      double precision FUNCTION EXP1(X)
      implicit double precision (a-h,o-z)
      double precision X
      double precision ZERO,NEG50
      DATA ZERO/0.0d0/,NEG50/-50.0d0/
C
      IF (.NOT.(X.LT.NEG50)) GO TO 10
      EXP1 = ZERO
      GO TO 20
 10   EXP1 = EXP(X)
```

```
 20   RETURN
      END
C
      SUBROUTINE ORDER (ALB,ITH)
C
C     Find order of absolute values of ALB.
C
      implicit double precision (a-h,o-z)
      double precision ALB(*)
      INTEGER ITH(*)
      INTEGER J,K,K1,ITHK
      double precision ALJ
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
C
      J = 1
 10   IF (.NOT.(J.LE.IR)) GO TO 60
      ALJ = ABS(ALB(J))
      K = J - 1
 20   IF (.NOT.(K.GT.0)) GO TO 40
      ITHK = ITH(K)
      K1 = K + 1
      IF (.NOT.(ALJ.GT.ABS(ALB(ITHK)))) GO TO 50
      ITH(K1) = ITHK
      GO TO 30
 30   K = K - 1
      GO TO 20
 40   K = 0
      K1 = 1
 50   ITH(K1) = J
      J = J + 1
      GO TO 10
 60   NDTSRT = .FALSE.
      RETURN
      END
C
      double precision FUNCTION ERRBD(N,ALB,ANC,UU,CX)
C
C     Find bound on tail probability using mgf.
C     Cut-off point returned to CX.
C
      implicit double precision (a-h,o-z)
      double precision U,UU,CX
      INTEGER N(*)
      double precision ALB(*),ANC(*)
      double precision SUM1,ALJ,ANCJ,X,Y,CONST
      INTEGER J,NJ
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA HALF/0.5d0/,ONE/1.0d0/,TWO/2.0d0/
C
      CALL GCNTR
      U = UU
      CONST = U*SIGSQ
      SUM1 = U*CONST
```

```
      U = TWO*U
      J = IR
   10 IF (.NOT.(J.GT.0)) GO TO 20
      NJ = N(J)
      ALJ = ALB(J)
      ANCJ = ANC(J)
      X = U*ALJ
      Y = ONE - X
      CONST = CONST + ALJ*(ANCJ/Y + NJ)/Y
      SUM1 = SUM1 + ANCJ*(X/Y)**2
      SUM1 = SUM1 + NJ*(X**2/Y + ALOG1(-X,.FALSE.))
      J = J - 1
      GO TO 10
   20 ERRBD = EXP1(-HALF*SUM1)
      CX = CONST
      RETURN
      END
C
      double precision FUNCTION CTFF(N,ALB,ANC,ACCX,UPN)
C
C     Find CTFF so that P(QF > CTFF) < ACCX if UPN > 0;
C     P(QF < CTFF) < ACCX otherwise.
C
      implicit double precision (a-h,o-z)
      double precision ACCX,UPN
      INTEGER N(*)
      double precision ALB(*),ANC(*)
      double precision U1,U2,U,RB,CONST,C1,C2
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0d0/,ONE/1.0d0/,TWO/2.0d0/
C
      F1(I) = U2/(ONE + U2*RB)
      F2(I) = (C1 - AMEAN)/(C2 - AMEAN)
C
      U2 = UPN
      U1 = ZERO
      C1 = AMEAN
      IF (.NOT.(U2.GT.ZERO)) GO TO 10
      RB = ALMAX
      GO TO 20
   10 RB = ALMIN
   20 RB = TWO*RB
      U = F1(0)
   30 IF (.NOT.(ERRBD(N,ALB,ANC,U,C2).GT.ACCX)) GO TO 40
      U1 = U2
      C1 = C2
      U2 = TWO*U2
      U = F1(0)
      GO TO 30
   40 U = F2(0)
   50 IF (.NOT.(U.LT.0.9)) GO TO 80
      U = (U1 + U2)/TWO
      IF (.NOT.(ERRBD(N,ALB,ANC,U/(ONE + U*RB),CONST).GT.ACCX)) GO TO 60
      U1 = U
      C1 = CONST
      GO TO 70
   60 U2 = U
      C2 = CONST
```

```
 70   U = F2(0)
      GO TO 50
 80   CTFF = C2
      UPN = U2
      RETURN
      END
C
      double precision FUNCTION TRUNCN(N,ALB,ANC,UU,TAUSQ)
C
C     Bound integration error due to truncation at U.
C
      implicit double precision (a-h,o-z)
      double precision U,UU,TAUSQ
      INTEGER N(*)
      double precision ALB(*),ANC(*)
      double precision SUM1,SUM2,PROD1,PROD2,PROD3,ALJ,ANCJ,X,Y,
     &ERR1,ERR2
      INTEGER J,NJ,NS
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,QUART,
     &HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0d0/,QUART/0.25d0/,HALF/0.5d0/,ONE/1.0d0/,TWO/2.0d0/
C
      CALL GCNTR
      U = UU
      SUM1 = ZERO
      PROD2 = ZERO
      PROD3 = ZERO
      NS = 0
      SUM2 = (SIGSQ + TAUSQ)*U**2
      PROD1 = TWO*SUM2
      U = TWO*U
      J = 1
 10   IF (.NOT.(J.LE.IR)) GO TO 40
      ALJ = ALB(J)
      ANCJ = ANC(J)
      NJ = N(J)
      X = (U*ALJ)**2
      SUM1 = SUM1 + ANCJ*X/(ONE + X)
      IF (.NOT.(X.GT.ONE)) GO TO 20
      PROD2 = PROD2 + NJ*LOG(X)
      PROD3 = PROD3 + NJ*ALOG1(X,.TRUE.)
      NS = NS + NJ
      GO TO 30
 20   PROD1 = PROD1 + NJ*ALOG1(X,.TRUE.)
 30   J = J + 1
      GO TO 10
 40   SUM1 = HALF*SUM1
      PROD2 = PROD1 + PROD2
      PROD3 = PROD1 + PROD3
      X = EXP1(-SUM1 - QUART*PROD2)/PI
      Y = EXP1(-SUM1 - QUART*PROD3)/PI
      IF (.NOT.(NS.EQ.0)) GO TO 50
      ERR1 = ONE
      GO TO 60
 50   ERR1 = X*TWO/NS
 60   IF (.NOT.(PROD3.GT.ONE)) GO TO 70
      ERR2 = 2.5d0*Y
      GO TO 80
```

```
 70   ERR2 = ONE
 80   IF (.NOT.(ERR2.LT.ERR1)) GO TO 90
      ERR1 = ERR2
 90   X = HALF*SUM2
      IF (.NOT.(X.LE.Y)) GO TO 100
      ERR2 = ONE
      GO TO 110
100   ERR2 = Y/X
110   IF (.NOT.(ERR1.LT.ERR2)) GO TO 120
      TRUNCN = ERR1
      GO TO 130
120   TRUNCN = ERR2
130   RETURN
      END
C
      SUBROUTINE FINDU (N,ALB,ANC,UTX,ACCX)
C
C     Find U such that TRUNCN(U) < ACCX & TRUNCN(U / 1.2) > ACCX.
C
      implicit double precision (a-h,o-z)
      double precision UTX,ACCX
      INTEGER N(*)
      double precision ALB(*),ANC(*)
      double precision U,UT
      double precision DIVIS(4)
      INTEGER I
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,FOUR
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA DIVIS/2.0d0,1.4d0,1.2d0,1.1d0/, FOUR/4.0d0/
C
      UT = UTX
      U = UT/FOUR
      IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).GT.ACCX)) GO TO 20
      U = UT
 10   IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).GT.ACCX)) GO TO 40
      UT = UT*FOUR
      U = UT
      GO TO 10
 20   UT = U
      U = U/FOUR
 30   IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).LE.ACCX)) GO TO 40
      UT = U
      U = U/FOUR
      GO TO 30
 40   DO 50 I = 1,4
      U = UT/DIVIS(I)
      IF (.NOT.(TRUNCN(N,ALB,ANC,U,ZERO).LE.ACCX)) GO TO 50
      UT = U
 50   CONTINUE
      UTX = UT
      RETURN
      END
C
      SUBROUTINE INTEGR (N,ALB,ANC,NTERM,AINTRV,TAUSQ,MAIN)
C
C     Carry out integration with NTERM terms, at interval AINTRV.
C     If not MAIN then multiply integrand by  1 - exp(-0.5 * TAUSQ *
C     U**2).
C
```

```
      implicit double precision (a-h,o-z)
      INTEGER NTERM
      double precision AINTRV,TAUSQ
      LOGICAL MAIN
      INTEGER N(*)
      double precision ALB(*),ANC(*)
      double precision AINPI,U,SUM1,SUM2,SUM3,X,Y,Z
      INTEGER K,J,NJ
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,QUART,
     &HALF,ONE,TWO
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA QUART/0.25d0/,HALF/0.5d0/,ONE/1.0d0/,TWO/2.0d0/
C
      AINPI = AINTRV/PI
      K = NTERM
 10   IF (.NOT.(K.GE.0)) GO TO 50
      U = (K + HALF)*AINTRV
      SUM1 = -TWO*U*C
      SUM2 = ABS(SUM1)
      SUM3 = -HALF*SIGSQ*U**2
      J = IR
 20   IF (.NOT.(J.GT.0)) GO TO 30
      NJ = N(J)
      X = TWO*ALB(J)*U
      Y = X**2
      SUM3 = SUM3 - QUART*NJ*ALOG1(Y,.TRUE.)
      Y = ANC(J)*X/(ONE + Y)
      Z = NJ*ATAN(X) + Y
      SUM1 = SUM1 + Z
      SUM2 = SUM2 + ABS(Z)
      SUM3 = SUM3 - HALF*X*Y
      J = J - 1
      GO TO 20
 30   X = AINPI*EXP1(SUM3)/U
      IF (.NOT.(.NOT.MAIN)) GO TO 40
      X = X*(ONE - EXP1(-HALF*TAUSQ*U**2))
 40   SUM1 = SIN(HALF*SUM1)*X
      SUM2 = HALF*SUM2*X
      AINTL = AINTL + SUM1
      ERSM = ERSM + SUM2
      K = K - 1
      GO TO 10
 50   RETURN
      END
C
      double precision FUNCTION CFE(N,ALB,ANC,ITH,X)
C
C     Coefficient of TAUSQ in error when convergence factor of
C     exp(-0.5 * TAUSQ * U**2) is used when df is evaluated at X.
C
      implicit double precision (a-h,o-z)
      double precision X
      INTEGER N(*),ITH(*)
      double precision ALB(*),ANC(*)
      double precision AXL,AXL1,AXL2,SXL,SUM1,ALJ
      INTEGER J,K,IT,ITK
      DOUBLE PRECISION AINTL,ERSM
      double precision PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,ZERO,ONE,
     &FOUR,HUNDRD
```

```
      INTEGER ICOUNT,IR,LIM
      LOGICAL NDTSRT,FAIL
      COMMON /QFCOM/ AINTL,ERSM,PI,ALN28,SIGSQ,ALMAX,ALMIN,AMEAN,C,
     1  ICOUNT,IR,NDTSRT,FAIL,LIM
      DATA ZERO/0.0d0/,ONE/1.0d0/,FOUR/4.0d0/,HUNDRD/100.0d0/
C
      CALL GCNTR
      IF (.NOT.(NDTSRT)) GO TO 10
      CALL ORDER (ALB,ITH)
 10   AXL = ABS(X)
      SXL = SIGN(ONE,X)
      SUM1 = ZERO
      J = IR
 20   IF (.NOT.(J.GT.0)) GO TO 70
      IT = ITH(J)
      IF (.NOT.(ALB(IT)*SXL.GT.ZERO)) GO TO 60
      ALJ = ABS(ALB(IT))
      AXL1 = AXL - ALJ*(N(IT) + ANC(IT))
      AXL2 = ALJ/ALN28
      IF (.NOT.(AXL1.GT.AXL2)) GO TO 30
      AXL = AXL1
      GO TO 60
 30   IF (.NOT.(AXL.GT.AXL2)) GO TO 40
      AXL = AXL2
 40   SUM1 = (AXL - AXL1)/ALJ
      K = J - 1
 50   IF (.NOT.(K.GT.0)) GO TO 70
      ITK = ITH(K)
      SUM1 = SUM1 + (N(ITK) + ANC(ITK))
      K = K - 1
      GO TO 50
 60   J = J - 1
      GO TO 20
 70   IF (.NOT.(SUM1.GT.HUNDRD)) GO TO 80
      CFE = ONE
      FAIL = .TRUE.
      GO TO 90
 80   CFE = 2**(SUM1/FOUR)/(PI*AXL**2)
 90   RETURN
      END
```